PROGETTO RETI DI CALCOLATORI

Antonio Prisco mat.0124002678

1 Descrizione del progetto

Il progetto proposto prevede lo sviluppo di un'applicazione client/server parallelo per semplificare la gestione degli esami universitari. La segreteria dell'università avrà la capacità di inserire nuovi esami nel server, consentendo la memorizzazione di tali dati in memoria. Inoltre, sarà in grado di inoltrare le richieste di prenotazione degli studenti al server universitario.

Dall'altra parte, gli studenti potranno consultare la segreteria per verificare la disponibilità degli esami relativi a un determinato corso. In seguito, potranno inviare richieste di prenotazione per gli esami desiderati. Il server universitario sarà responsabile di ricevere sia le aggiunte di nuovi esami che le prenotazioni degli studenti.

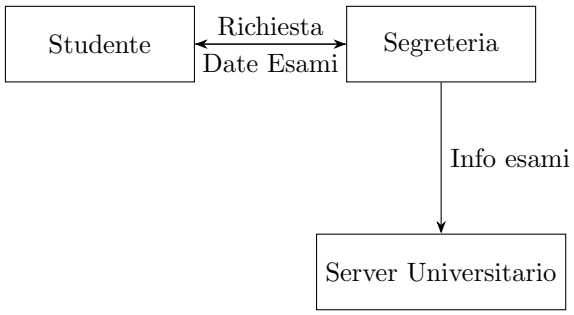
L'implementazione di questo sistema agevola il flusso di informazioni tra la segreteria e gli studenti, semplificando la gestione degli esami universitari e migliorando l'efficienza complessiva del processo.

2 Descrizione e schema dell'architettura

L'architettura del progetto è rappresentata da un diagramma che coinvolge tre componenti principali: lo studente, la segreteria e il server universitario.

Studente:Rappresenta l'utente finale, lo studente, che

interagisce con il sistema. Può inviare richieste alla segreteria per ottenere informazioni sugli esami disponibili. Segreteria:La segreteria è responsabile di gestire le richieste dello studente e fornire informazioni sugli esami disponibili. Risponde alle richieste degli studenti inviando loro le date degli esami. Server Universitario: Il server universitario è una componente centrale che gestisce le richieste della segreteria ,coinvolto nella gestione del database delle informazioni sugli esami. Il diagramma include frecce direzionali che rappresentano le interazioni tra gli elementi dell'architettura. Lo studente invia richieste alla segreteria, che a sua volta interagisce con il server universitario per ottenere le informazioni necessarie. La segreteria fornisce poi le risposte allo studente, inclusa l'informazione sulle date degli esami.



3 Dettagli implementativi dei client/server

Il software è stato implementato per consentire agli studenti di accedere a due server distinti: il server universitario e il server della segreteria. Il programma è scritto in linguaggio C e utilizza il protocollo TCP/IP per la comunicazione tra il client e i server.

3.1 Interfaccia Utente

Una volta connesso con successo ai server, il Client presenta un menu utente che consente agli studenti di effettuare varie operazioni:

1. **Verifica Disponibilità Esami per un Corso**: Invia una richiesta al server della segreteria per verificare la disponibilità di esami per un determinato corso.
2. **Richiesta Prenotazione Esame**: Permette agli studenti di inviare una richiesta di prenotazione di un esame al server della segreteria.
3. **Chiusura Programma**: Permette agli studenti di chiudere il programma.

Dopo aver inviato una richiesta, il client attende una risposta dal server corrispondente. Se la richiesta viene elaborata con successo, viene stampato un messaggio di conferma. In caso di errore durante la comunicazione con il server, viene stampato un messaggio di errore appropriato. Quando l'utente sceglie di chiudere il programma, viene inviata una richiesta di chiusura,successivamente, il socket viene chiuso.

3.2 Server Universitario

Viene definita una struttura Esame per memorizzare le informazioni sugli esami, con campi per il corso e la data.

La funzione scriviEsameSuFile è utilizzata per aggiungere un esame su un file.Se l'utente sceglie di aggiungere un esame, il programma richiede il nome del corso e la data, li salva nella struttura Esame e li scrive su un file chiamato "esami.txt".

La funzione riceviRichiestaUniversitario è responsabile della ricezione delle richieste dal client. Crea un socket, lo associa a una porta e si mette in ascolto. Quando arriva una connessione, legge la richiesta dal client e la restituisce.

3.3 Server Segreteria

La funzione leggiEsamiDaFile legge gli esami da un file di testo e li memorizza in un array di strutture Esame. Restituisce un valore booleano per indicare se l'operazione è riuscita.

La funzione inoltraRichiestaUniversitario si occupa di connettersi al server universitario e inoltrare una richiesta ad esso.

Nel main, viene creato un socket per gestire le richieste del client e si mette in ascolto per le connessioni in entrata.

Quando arriva una connessione dal client, legge il tipo di richiesta e agisce di conseguenza:

Se la richiesta è per ottenere la lista degli esami per un determinato corso, legge gli esami dal file, filtra quelli relativi al corso desiderato e li mostra.

Se la richiesta è per inoltrare una richiesta di prenotazione degli studenti al server universitario, inoltra la richiesta al server universitario tramite la funzione inoltraRichiestaUniversitario.

Se la richiesta è per chiudere il server, imposta una variabile booleana per indicare che il server deve terminare.

Se la richiesta è diversa da quelle gestite, stampa un messaggio di errore.

4 Parti rilevanti del codice sviluppato

Client "Studente"

Contiene le funzioni per connettersi ai server tramite indirizzi IP e porte.

```c
// Funzione per la connessione al server
int connettiServer(const char *server_ip, int port) {
    int sock;
    struct sockaddr_in server_addr;

    // Creazione del socket del client studente
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Errore nella creazione del socket del client studente");
        exit(EXIT_FAILURE);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(port);
    server_addr.sin_addr.s_addr = inet_addr(server_ip);

    // Connessione al server
    if (connect(sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Errore nella connessione al server");
        exit(EXIT_FAILURE);
    }

    return sock;
}

int connettiServerSegreteria() {
    int sock;
    struct sockaddr_in server_addr;

    // Creazione del socket del client studente
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Errore nella creazione del socket del client studente");
        exit(EXIT_FAILURE);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT_SEGRETERIA);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1"); // IP del server segreteria

    // Connessione al server segreteria
    if (connect(sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Errore nella connessione al server segreteria");
        exit(EXIT_FAILURE);
    }
```

Successivamente c'è il blocco principale della funzione main() del Client. Qui avviene l'interazione con l'utente attraverso un menu e vengono prese le azioni appropriate in base alla scelta dell'utente. Vengono chiamate le funzioni definite in precedenza per connettersi ai server.

```c
int main() {
    // Configurazione per il server universitario
    const char *server_universitario_ip = "127.0.0.1";

    // Connessione al server universitario
    int client_studente_socket_universitario = connettiServer(server_universitario_ip, PORT_UNIVERSITARIO);
    int client_studente_socket_segreteria = connettiServerSegreteria();
    // Menù utente
    int scelta;
    printf("1. Verifica disponibilità esami per un corso\n");
    printf("2. Richiesta prenotazione esame\n");
    printf("3. Chiudi menù.\n");
    printf("Scelta: ");
    scanf("%d", &scelta);

    if (scelta == 1) {
        // Invia al server segreteria il tipo di richiesta per la disponibilità di esami (ad esempio, 1)
        int tipo_richiesta = 1;
        send(client_studente_socket_segreteria, &tipo_richiesta, sizeof(tipo_richiesta), 0);
        printf("Richiesta di disponibilità inviata al server segreteria.\n");
    }
    else if (scelta == 2) {
        // Richiesta al server segreteria per la prenotazione di un esame
        char corso_prenotazione[50];
        printf("Inserisci il corso per la prenotazione: ");
        scanf("%49s", corso_prenotazione);

        // Invia la richiesta di prenotazione al server segreteria
        inoltraRichiestaPrenotazioneSegreteria(client_studente_socket_segreteria, scelta);

        // Invia il nome del corso al server segreteria
        send(client_studente_socket_segreteria, corso_prenotazione, strlen(corso_prenotazione), 0);
        printf("Richiesta di prenotazione inviata al server segreteria.\n");
    }
    else if (scelta == 3) {
        // Invia al server segreteria il tipo di richiesta per la chiusura (ad esempio, 3)
        int tipo_richiesta = 3;
        send(client_studente_socket_segreteria, &tipo_richiesta, sizeof(tipo_richiesta), 0);

        // Chiudi il socket dopo aver inviato la richiesta di chiusura
        close(client_studente_socket_segreteria);
        printf("Chiusura del programma.\n");
    }
    else {
        printf("Scelta non valida. Riprova.\n");
```

Server Segreteria

Nel server segreteria vengono definite le funzioni "leggiesamedafile" e "inoltraRichiestaUniversitario" e poi successivamente chiamate nel main riportato qui in basso.

All'interno del main:

Creazione del socket: In questa parte, il server crea un socket utilizzando la funzione socket() con AF INET e SOCK STREAM per indicare la famiglia di indirizzi (IPv4) e il tipo di socket (TCP), rispettivamente.

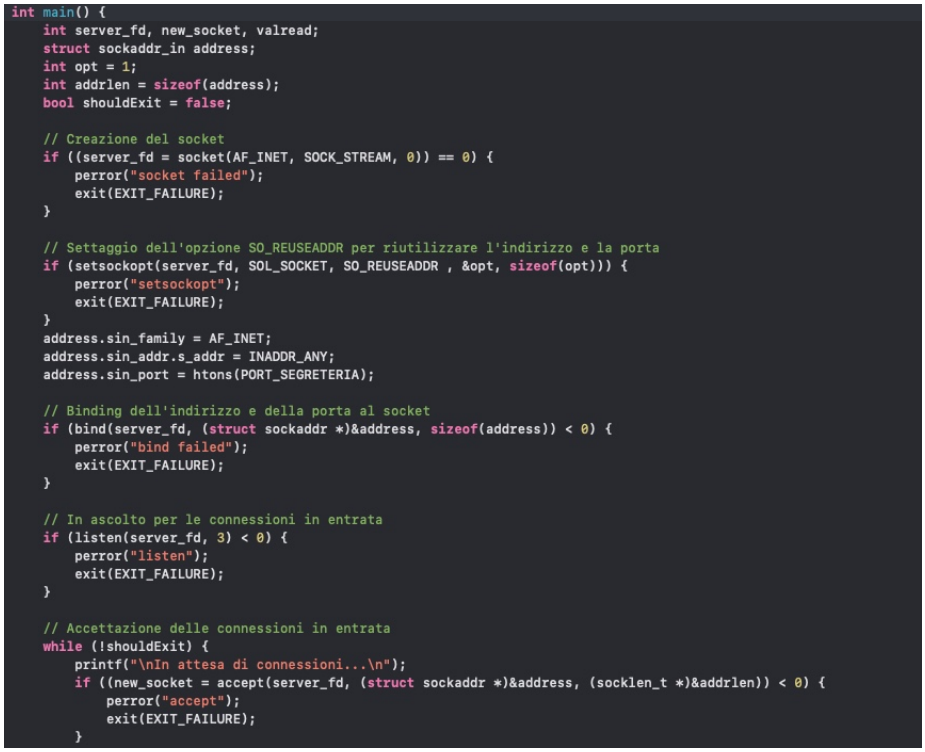
Il server entra in modalità di ascolto utilizzando la funzione listen() per accettare connessioni in arrivo.

Accettazione delle connessioni in entrata: Questa parte del codice utilizza la funzione accept() per accettare con-

nessioni in arrivo dai client e inizia a gestire le richieste.

Gestione della richiesta: Il server legge il tipo di richiesta inviata dal client e gestisce le diverse richieste tramite un blocco switch. Attualmente, il server gestisce le richieste per ottenere gli esami per un corso specifico, inoltrare richieste di prenotazione degli studenti e chiudere la connessione.

Queste parti sono fondamentali per l'avvio del server, l'accettazione delle connessioni dei client e la gestione delle richieste in arrivo.



```
int main() {
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    bool shouldExit = false;

    // Creazione del socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // Settaggio dell'opzione SO_REUSEADDR per riutilizzare l'indirizzo e la porta
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR , &opt, sizeof(opt))) {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT_SEGRETERIA);

    // Binding dell'indirizzo e della porta al socket
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    // In ascolto per le connessioni in entrata
    if (listen(server_fd, 3) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }

    // Accettazione delle connessioni in entrata
    while (!shouldExit) {
        printf("\nIn attesa di connessioni...\n");
        if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen)) < 0) {
            perror("accept");
            exit(EXIT_FAILURE);
        }
```

```c
int main() {
        // Gestione della richiesta
        switch (richiesta) {
            case 1: {
                Esame listaEsami[MAX_ESAMI];
                int numEsami = 0;

                if (!leggiEsamiDaFile("esami.txt", listaEsami, &numEsami)) {
                    fprintf(stderr, "Errore durante la lettura degli esami dal file.\n");
                    exit(EXIT_FAILURE);
                }

                // Richiedi all'utente il nome del corso
                char corsoDesiderato[50];
                printf("Inserisci il nome del corso: ");
                scanf("%s", corsoDesiderato);

                // Mostra solo gli esami relativi al corso desiderato
                printf("Esami per il corso '%s':\n", corsoDesiderato);
                int esamiTrovati = 0;
                for (int i = 0; i < numEsami; i++) {
                    if (strcmp(listaEsami[i].corso, corsoDesiderato) == 0) {
                        printf("%s %s\n", listaEsami[i].corso, listaEsami[i].data);
                        esamiTrovati++;
                    }
                }

                if (esamiTrovati == 0) {
                    printf("Nessun esame trovato per il corso '%s'.\n", corsoDesiderato);
                }
                break;
            }
            case 2: {
                // Inoltra la richiesta di prenotazione degli studenti al server universitario
                printf("Inoltro della richiesta di prenotazione degli studenti al server universitario...\n");
                inoltraRichiestaUniversitario(richiesta);
                break;
            }
            case 3: {
                printf("Chiusura del server...\n");
                shouldExit = true;
                break;
            }
            default:
                printf("Richiesta non valida.\n");
```

Server Universitario

Nel server universitario vengono definite le funzioni "ScriviEsameSuFile" e "RiceviRichiestaUniversitario" e poi successivamente chiamate nel main riportato qui in basso.

All'interno del main:

Viene creato un socket per gestire le connessioni dei client utilizzando la funzione socket(). Il socket è di tipo AF INET per la comunicazione su una rete IPv4 e di tipo SOCK STREAM per una comunicazione orientata alla connessione.

Il server si mette in ascolto per le connessioni in entrata utilizzando la funzione listen(). Questo passaggio consente al server di accettare le connessioni in arrivo dai

client.

Il server chiede all'utente se desidera aggiungere un nuovo esame. Se la risposta è "si", il server chiede il nome del corso e la data dell'esame all'utente utilizzando scanf(), li legge nella struttura nuovoEsame e li scrive su un file chiamato "esami.txt" utilizzando la funzione scriviEsameSuFile().

```c
int main() {
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;

    int addrlen = sizeof(address);
    Esame nuovoEsame;

    // Creazione del socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // Settaggio dell'opzione SO_REUSEADDR per riutilizzare l'indirizzo e la porta
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR , &opt, sizeof(opt))) {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    // Binding dell'indirizzo e della porta al socket
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    // In ascolto per le connessioni in entrata
    if (listen(server_fd, 3) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }

    // Creazione della struct per gli esami e inizializzazione del numero di esami

    printf("Vuoi aggiungere un esame? (si/no): ");
    char risposta[3];
    scanf("%s", risposta);
    if (strcmp(risposta, "si") == 0) {
        printf("Inserisci il nome del corso: ");
            scanf("%s", nuovoEsame.corso);

            printf("Inserisci la data dell'esame: ");
            scanf("%s", nuovoEsame.data);
```

5 Manuale utente con le istruzioni su compilazione ed esecuzione

Istruzioni per la compilazione:

1. Presa la directory di Server Univ: `gcc main.c -o`

`main`

2. Presa la directory di Server Seg: `gcc main.c -o main`
3. Presa la directory di Client Stud: `gcc main.c -o main`

Istruzioni per l'esecuzione:

1. Avviare Server Univ che ti chiederà se vuoi aggiungere una data di esame con il relativo nome del corso e la sua data;
2. Avviare Server Seg che attenderà la richiesta dal Client;
3. Avviare Client Stud che ti mostrerà un menu con tre scelte di richieste differenti da selezionare.