Università degli studi di Napoli Parthenope
Laboratorio di Reti di Calcolatori
Progetto: Università

Davide Aiello 0124002594

1 Descrizione

L'applicazione client/server parallelo per gestire gli esami universitari si propone di automatizzare e semplificare il processo di prenotazione e gestione degli esami. La segreteria, come interfaccia principale, inserisce dettagli sugli esami nel server dell'università, consentendo di salvare i dati su file. Inoltre, gestisce le richieste di prenotazione degli studenti, inoltrando tali richieste al server universitario.

Gli studenti possono interrogare la segreteria per verificare la disponibilità di esami per un determinato corso e inviare richieste di prenotazione. Il server universitario riceve le nuove informazioni sugli esami e le prenotazioni degli studenti, assegnando a ciascuna richiesta un numero progressivo. Questo numero viene inviato alla segreteria dal server universitario e successivamente trasmesso agli studenti, garantendo un tracciamento sistematico delle prenotazioni.

La struttura parallela del sistema consente una gestione efficiente e concorrente delle richieste, migliorando la scalabilità e la reattività del sistema. L'utilizzo di un server centrale favorisce la coerenza dei dati e semplifica la manutenzione. Complessivamente, il progetto mira a ottimizzare il processo di gestione degli esami universitari, garantendo un accesso agevole alle informazioni e una registrazione accurata delle prenotazioni degli studenti.

2 Descrizione e schema dell'architettura

Figura 1: Architettura della rete

Richiesta esami:

- Studente si connette a Segreteria e chiede se ci siano esami disponibili per un corso
- La segreteria riceve il nome del corso e lo inoltra al Server Universitario
- Il Server Universitario riceve il nome del corso e controlla se ci siano esami disponibili per il corso ricevuto
- Il Server inoltra la lista degli esami disponibili alla segreteria
- La Segreteria inoltra la lista degli esami disponibili allo studente che ha effettuato la richiesta

Aggiunta esami:

- Segreteria inoltra al sever il nome del corso, la data di esame e il numero di CFU dell'esame da aggiungere.
- Il Server Universitario riceve la richiesta e salva sul file 'examsList' l'esame.

Prenotazione esame:

- Una volta ricevuta dalla Segreteria la lista degli esami disponibili per il corso richiesto lo studente può decidere a quale prenotarsi inserendo il codice dell'esame e la sua matricola
- Lo studente inoltra alla Segreteria l'esame scelto e la marticola
- La Segreteria riceve l'esame e la matricola dello studente e l'inoltra al Server Universitario
- Il Server Universitario riceve esame e matricola e salva la prenotazione sul file di prenotazione dell'esame scelto associando a quest'ultima un Id univoco di prenotazione
- Il Server Universitario inoltra alla Segreteria il numero di prenotazione

3 Dettagli implementativi dei client/server

Il client scrive sul socket di connessione alla segreteria la struct Request che contiene:

- requestType: tipo di richiesta effettuata
  - requestType = 1: indica che lo studente richiede la lista delgi esami disponibili
- Exam: struct Exam che contiente:
  - name: nome del corso
  - data: data del corso
  - cfu: CFU del corso

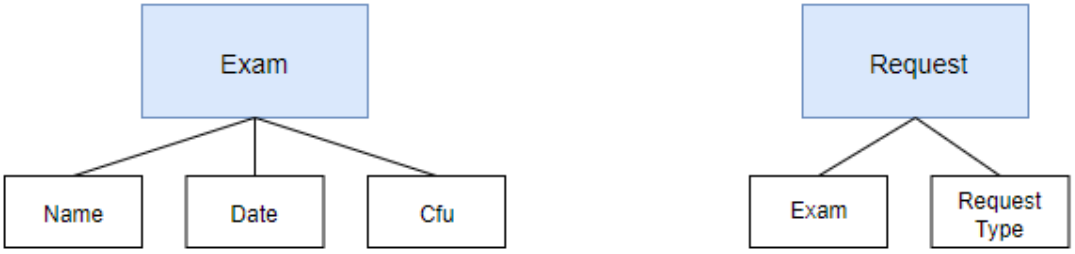

Figura 2: Dati inviati alla segreteria da parte dello studente

```
struct Exam {
    char name[40];
    char date[25];
    short cfu;
};

struct Request {
    short requestType;
    struct Exam exam;
};
```

Lo studente inserisce il nome del corso che viene poi salvato nella struct availableExamRequest.exam.name.
Il campo requestType viene settato a 1.
Dopodichè viene creato il socket 'socketFD' e si effettua la connessione alla Segreteria.
Si scrive la struct availableExamRequest sul socket inviando così la richiesta alla segreteria.
La segreteria effettua una read sul socket connFD, se requestType == 1 allora si collega al server universitario, salva sulla struct retriveAvailableExamRequest l'esame ricevuto dallo studente e setta requestType a 2.
Esegue una write sul socket di connessione con il server universitario della struct e quindi invia al server la richiesta.
Il server esegue una read sul socket connFD, se requestType == 2, allora apre il file 'examList' e controlla quali esami salvati corrispondono al corso richiesto, salvando ogni esame disponibile su un array di struct Exam.
Successivamente scrive sul socket di connessione con la segreteria l'intero array con gli esami disponibili. La segreteria riceve l'array di struct e lo inoltra allo studente che ha effettuato la richiesta che può ora visualizzare le date di esame per il corso richiesto.

Dopo aver ricevuto la lista degli esami disponibili per il corso richiesto, lo studente inserisce il codice dell'esame a cui desidera iscriversi e la propria matricola. Codice e matricola vengono scritti sul socket di connessione con la segreteria socketFD, la segreteria riceve ed inoltra al server universitario quest'ultimi.

Il Server riceve il codice dell'esame e apre il file di prenotazioni che corrisponde all'esame richiesto ed aggiunge in coda al file la prenotazione dell'esame con il numero assegnatogli e la matricola.

4 Parti rilevanti del codice

Richiesta da parte dello studetente della lista esami disponibili per un determinato corso:

```
availableExamRequest = getExamName(&availableExamRequest);

if (availableExamRequest.exam.name[0] != '\000') {

    //crea il socket per la comunicazione con la segreteria
    socketFD = connectToSegreteria(socketFD, &serverSegreteriaAddress);

    //inoltra alla segreteria l'esame richiesto
    sendRequestedExamToSegreteria(socketFD, &availableExamRequest);
```

La Segreteria invia la server la richiesta ricevuta dallo studente:

```
void retriveAvailableExams(struct Exam exam, int connFD) {
    int socketFD;
    struct sockaddr_in serverUniAddress;
    struct Request retriveAvailableExamRequest;

    retriveAvailableExamRequest.requestType = 2;
    retriveAvailableExamRequest.exam = exam;

    //crea il socket per la comunicazione con il server università
    socketFD = connectToUniversita(socketFD, &serverUniAddress);

    //inoltra al server l'esame che l'utente ha richiesto
    sendRequestedExamToServer(socketFD, retriveAvailableExamRequest);
    struct Exam *recivedExam = getAndSendExams(connFD, socketFD);

    struct Exam examToBook;
    getAndSendExamToBook(connFD, socketFD, &examToBook);

    getAndSendMatricola(connFD, socketFD);

    getAndSendBookingId(connFD, socketFD);

    //chiude la connsessione con il server
    close( fd: socketFD);
    free( ptr: recivedExam);
}
```

Il server Universitario invia alla Segreteria la lista degli esami disponibili per il corso richiesto:

```
else if (recivedRequest.requestType == 2) {
    struct Exam *examsList;
    struct Exam *examsSelectedByName;
    int examsSelectedByNameCount;
    retriveExamList(&examsList);
    examsSelectedByNameCount = selectExamsByName(examsList, examName: recivedRequest.exam.name, &examsSelectedByName);
    sendSelectedExamsToSegreteria(connFD, &examsSelectedByName, examsSelectedByNameCount);
```

La segreteria invia al server un nuovo esame da aggiungere:

```
void sendAddedExamToServer(struct Exam exam) {
    int socketFD;
    struct sockaddr_in serverUniAddress;
    struct Request addExamRequest;

    addExamRequest.requestType = 1;
    addExamRequest.exam = exam;
    socketFD = connectToUniversita(socketFD, &serverUniAddress);

    //scrive su socketFD la struct addExamRequest che contiene l'esame da aggiungere
    if (write( fd: socketFD, buf: &addExamRequest, n: sizeof(addExamRequest)) != sizeof(addExamRequest)) {
        perror( s: "Write error 7");
        exit( status: 1);
    }
    printf( format: "Esame aggiunto con successo!");

    //chiude la connsessione con il server
    close( fd: socketFD);
}
```

Il server aggiunge l'esame ricevuto sul file "ExamList":

```
//aggiunge l'esame exam in coda al file examList
void addExamToFile(struct Exam exam) {
    FILE *examListFD;
    if ((examListFD = fopen( filename: "examList", modes: "a+")) == NULL) {
        printf( format: "error opening the file");
        exit( status: 1);
    }

    fwrite( ptr: &exam, size: sizeof(struct Exam), n: 1, s: examListFD);
    fclose( stream: examListFD);
    printf( format: "Esame aggiunto con successo!\n");
}
```

5 Manuale

5.1 Istruzioni di compilazione:

in progetto-universita/Wrapper:

- gcc -c -o wrapper.o wrapper.c

in progetto-universita/universita:

- gcc -c -o universita.o -I../Wrapper universita.c
- gcc -o Universita universita.o ../Wrapper/wrapper.o

in progetto-universita/segreteria:

- gcc -c -o segreteria.o -I../Wrapper segreteria.c
- gcc -o Segreteria segreteria.o ../Wrapper/wrapper.o

in progetto-universita/studente:

- gcc -c -o studente.o -I../Wrapper studente.c
- gcc -o Studente studente.o ../Wrapper/wrapper.o

5.2 Istruzioni di esecuzione:

in progetto-universita/universita:

- ./Universita

in progetto-universita/segreteria:

- ./Segreteria

in progetto-universita/studente:

- ./Studente