Università
Università degli Studi di Napoli Parthenope, CdS in Informatica

Professore:
Di Nardo Emanuel

Candidato:
Bucciero Vincenzo

A.A. 2023/2024

Indice

Capitolo 1

Introduzione

Il mondo accademico, sempre in evoluzione, richiede soluzioni innovative per semplificare e ottimizzare processi critici come la gestione degli esami universitari. Nel contesto di questo imperativo, il mio progetto si propone di presentare un'applicazione client/server parallela concepita per rivoluzionare la gestione degli esami all'interno di un'istituzione accademica. In risposta alla traccia fornita, ho sviluppato un sistema completo, articolato e flessibile che coinvolge la Segreteria, gli Studenti e il Server Universitario.

Il progetto offre una solida base per migliorare l'efficienza e la trasparenza nel processo di prenotazione degli esami. Questa documentazione fornirà una panoramica dettagliata del lavoro, evidenziando aspetti cruciali come **l'architettura del sistema**, l'**implementazione** pratica di ciascun componente e le **scelte progettuali** adottate.

Da un punto di vista tecnico, ho utilizzato una struttura ***client/server parallela*** per garantire una comunicazione efficiente e senza intoppi tra i diversi **attori** coinvolti. L'obiettivo è stato non solo creare un'applicazione funzionale ma anche offrire un'esperienza utente fluida e intuitiva.

1.1 Descrizione del Progetto

Il progetto si propone di rispondere alle esigenze dinamiche e complesse della **gestione degli esami universitari** attraverso l'implementazione di un'applicazione **client/server parallela**. La centralità del processo si articola attorno a **tre attori principali**:

- La **Segreteria**;
- Gli **Studenti**;
- Il **Server Universitario**;

Vediamo nel dettaglio i tre attori:

- **Segreteria:** svolge un ruolo cruciale nel sistema, responsabile dell'inserimento degli esami nel server dell'università. Questa operazione può avvenire attraverso la conservazione dei dati in un file o in memoria, fornendo una flessibilità di implementazione. Inoltre, la Segreteria inoltra le richieste di prenotazione degli esami da parte degli studenti al Server Universitario e fornisce agli studenti le date disponibili per gli esami scelti;
- **Studenti:** interagiscono con il sistema per ottenere informazioni sugli esami disponibili e prenotare quelli di loro interesse. Possono chiedere alla Segreteria la disponibilità di esami per un determinato corso e inviare richieste di prenotazione. Questa interazione diretta tra gli Studenti e la Segreteria contribuisce a semplificare il processo di prenotazione e fornisce agli studenti un controllo attivo sulla loro partecipazione agli esami;
- **Server Universitario:** il cuore del sistema risiede nel Server Universitario, che riceve le nuove informazioni sugli esami aggiunti dalla Segreteria e le richieste di prenotazione degli Studenti. La comunicazione bidirezionale tra la Segreteria e il Server Universitario garantisce una gestione fluida delle operazioni.

Attraverso questa struttura, si mira a creare un sistema robusto e efficiente che semplifichi la complessità della **gestione degli esami universitari**. La modularità del progetto consente una facile estensione delle funzionalità e la sua scalabilità, rendendolo adatto a soddisfare le esigenze di istituzioni accademiche di diverse dimensioni e complessità.

1.2 Architettura del Sistema

Vediamo gli attori coinvolti che possono eseguire determinate funzioni:

- **Studenti:**
  - Chiede alla segreteria se ci siano esami disponibili per un corso.
  - Invia una richiesta di prenotazione di un esame alla segreteria.
- **Segreteria:**
  - Inserisce gli esami sul server dell'università (salvare in un file o conservare in memoria il dato).
  - Inoltra la richiesta di prenotazione degli studenti al server universitario.
  - Fornisce allo studente le date degli esami disponibili per l'esame scelto dallo studente.
- **Server Universitario:**
  - Riceve l'aggiunta di nuovi esami.
  - Riceve la prenotazione di un esame.

Capitolo 2

Implementazione Client/Server

2.1 Server Universitario

Il server universitario implementa una funzionalità cruciale per la sua operatività, ovvero il caricamento degli esami da un file di testo al momento dell'avvio. Questa procedura fornisce una base di dati iniziale che contribuisce in modo significativo all'efficienza e alla coerenza del sistema.

2.1.1 Caricamento date esami su file

Durante l'inizializzazione, il server apre il file "*exams.txt*" in modalità di lettura ("r"). Utilizzando la funzione ***fscanf***, il server estrae le informazioni sugli esami dal file, popolando la struttura dati ***exams[ ]***. L'iterazione continua finché non vengono lette tutte le informazioni o raggiunto il limite massimo di esami (MAX_EXAMS).

```
1  void load_exams_from_file() {
2      FILE *file = fopen("exams.txt", "r");
3      if (file == NULL) {
4          perror("Error opening exams file");
5          exit(EXIT_FAILURE);
6      }
7      num_exams = 0;
8      while (fscanf(file, "%49s %19s", exams[num_exams].course, exams
       [num_exams].exam_date) == 2) {
9          num_exams++;
10         if (num_exams >= MAX_DATE) {
11             break;
12         }
13     }
14
15     fclose(file);
16 }
```

La gestione degli errori è integrata per affrontare situazioni in cui il file non può essere aperto. Limiti come MAX_EXAMS aiutano a prevenire sovraccarichi eccessivi di memoria e garantiscono una gestione controllata delle risorse.

2.1.2 Aggiunta date esami

```
void add_exam(SOCKET client_socket, const char* course, const char*
     date) {
    if (num_exams < MAX_DATE) {
        strcpy(exams[num_exams].course, course);
        strcpy(exams[num_exams].exam_date, date);
    }

    handle_exam_add(client_socket);
}
```

Questa funzione aggiunge un nuovo esame alla lista exams quando richiamata dal client. Prende come argomenti il socket del client (client_socket), il corso (course), e la data dell'esame (date). Controlla se c'è spazio sufficiente nella lista degli esami (num_exams < MAX_EXAMS) e, in caso affermativo, copia il corso e la data dell'esame nella posizione corrispondente nella lista. Successivamente, chiama la funzione handle_exam_add(client_socket) per gestire l'aggiunta dell'esame.

2.1.3 Gestione delle richieste d'esame

```
void handle_exam_request(SOCKET client_socket, char* course) {
    // Send available exam dates to the client
    load_exams_from_file();
    char exam_dates[500] = "";
    for (int i = 0; i < num_exams; ++i) {
        if(strcmp(exams[i].course,course) == 0){
            strcat(exam_dates, exams[i].course);
            strcat(exam_dates, ": ");
            strcat(exam_dates, exams[i].exam_date);
            strcat(exam_dates, "\n");
        }
    }
    write(client_socket, exam_dates, strlen(exam_dates));
}
```

Questa funzione gestisce la richiesta da parte dello studente delle date degli esami per un determinato corso. Utilizza un ciclo for per scorrere la lista degli esami e, se il corso coincide con quello richiesto dallo studente, aggiunge le informazioni riguardanti il corso e la data alla stringa exam_dates. Infine, usa la funzione write per inviare la stringa al client.

2.1.4 Gestione delle prenotazioni all'esame

```
void handle_exam_reservation(SOCKET client_socket, const char* course, const char* date) {
    int found = 0;
    char buffer[100];
    load_reservation_from_file();

    for(int i = 0; i < MAX_DATE && found == 0; i++){
        if(!strcmp(course, exams[i].course)){
            if(!strcmp(date, exams[i].exam_date)){
                found = 1;
            }
        }
    }
    if(found == 1){

        FILE *reservation_file = fopen("reservations.txt", "w");
        if (reservation_file == NULL) {
            perror("\nError opening reservations file");
            return;
        }
        fclose(reservation_file);

        reservation_file = fopen("reservations.txt", "a");

        int i;
        for(i = 0; i < reservation_index; i++)
            if(!strcmp(course, reservation_data[i].name))
                break;

        if(i >= reservation_index){
            strcpy(reservation_data[i].name, course);
            reservation_data[i].reservation_num = 1;
            reservation_index++;
        }
        else
            reservation_data[i].reservation_num++;


        for(int j = 0; j < reservation_index; j++){
            printf("Index %d\n", reservation_index);
            printf("Name %s, Num %d\n", reservation_data[j].name, reservation_data[j].reservation_num);
            fprintf(reservation_file, "%s %d\n", reservation_data[j].name, reservation_data[j].reservation_num);
        }


        fclose(reservation_file);

        snprintf(buffer,sizeof(buffer),"reservation number %d \n\n", reservation_data[i].reservation_num);
        printf("Sending %lu",strlen(buffer));
        FullWrite(client_socket,buffer, strlen(buffer));
    }
    else{
```

```
52
53          snprintf(buffer, sizeof(buffer),"\nNot found exam  %s for
        date %s\n", course, date);
54          printf("Sending %lu",strlen(buffer));
55          FullWrite(client_socket,buffer, strlen(buffer));
56      }
57  }
```

Questa funzione gestisce la prenotazione simulata di un esame da parte dello studente. Crea un file di prenotazioni (reservations.txt), invoca la funzione kill per inviare un segnale SIGUSR1 al processo padre (indicando che una nuova prenotazione è stata effettuata), scrive le informazioni sulla prenotazione nel file e invia un messaggio al client confermando la prenotazione con un numero progressivo.

2.1.5 Gestione dell'aggiunta esame

```
1   void handle_exam_add(SOCKET client_socket){
2
3       // Add exam to the file
4       FILE *file = fopen("exams.txt", "a");
5       if (file == NULL) {
6           perror("Error opening exams file");
7           exit(EXIT_FAILURE);
8       }
9
10      // Check if there is enough space to add an exam
11      if (num_exams >= MAX_DATE) {
12          fprintf(stderr, "\nError: Maximum number of exams reached\n
        ");
13          fclose(file);
14          exit(EXIT_FAILURE);
15      }
16
17      // Write the new exam to the file
18      fprintf(file, "%s %s\n", exams[num_exams].course, exams[
        num_exams].exam_date);
19
20      // Increment the number of exams and close the file
21      load_exams_from_file();
22
23      write(client_socket, "\nExam added successfully!\0", 30);
24
25      fclose(file);
26  }
```

Questa funzione gestisce l'aggiunta di un nuovo esame al file exams.txt. Apre il file in modalità append ("a"), verifica se c'è spazio sufficiente, scrive il nuovo esame nel file, incrementa il contatore degli esami (num_exams), invia un messaggio di conferma al client e chiude il file.

2.2 Segreteria

La Segreteria è il componente del sistema responsabile della gestione degli esami universitari. La sua funzione principale è interagire con gli studenti, ricevere richieste di informazioni sugli esami, inoltrare prenotazioni al Server Universitario e fornire date degli esami ai singoli studenti. Vedremo nel dettaglio la descrizione delle principali attività svolte dalla Segreteria.

2.2.1 Inserimento degli Esami

La funzione add_exam consente all'utente di inserire nuovi esami specificando il corso e la data. Questa funzione rappresenta l'implementazione dell'inserimento degli esami.

```
1  void add_exam(const char *course, const char *date)
2  {
3      // Forward the request to add an exam to the university server
4      SOCKET client_socket = socket(AF_INET, SOCK_STREAM, 0);
5
6      struct sockaddr_in server_address;
7      server_address.sin_family = AF_INET;
8      server_address.sin_addr.s_addr = inet_addr(SERVER_IP);
9      server_address.sin_port = htons(SERVER_PORT);
10
11     if (connect(client_socket, (struct sockaddr *)&server_address,
       sizeof(server_address)) < 0)
12     {
13         perror("Error connecting to the university server");
14         exit(EXIT_FAILURE);
15     }
16
17     char request_type[] = "ADD_EXAM";
18     int byteRead = 0;
19
20     FullWrite(client_socket, request_type, sizeof(request_type));
21     sleep(3);
22
23     FullWrite(client_socket, course, strlen(course));
24     sleep(3);
25
26     FullWrite(client_socket, date, strlen(date));
27
28     // Receive confirmation of successful addition
29     char confirmation[100];
30     byteRead = read(client_socket, confirmation, sizeof(
       confirmation));
31
32     if (byteRead == 0)
33     {
34         printf("No exam added\n");
35     }
36     else
37     {
38         confirmation[byteRead] = '\0';
39         printf("Added Exam %s: on %s\n", course, date);
```

```
    }

    close(client_socket);
}
```

2.2.2 Inoltro delle Richieste degli Studenti

La funzione forward_exam_reservation gestisce l'inoltro delle richieste di prenotazione degli studenti al Server Universitario. Questa funzione è responsabile della comunicazione tra la Segreteria e il Server Universitario.

```
void forward_exam_reservation(int student_socket, const char *course, const char *date)
{
    // Forward the reservation request to the university server
    SOCKET client_socket = socket(AF_INET, SOCK_STREAM, 0);

    struct sockaddr_in server_address;
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = inet_addr(SERVER_IP);
    server_address.sin_port = htons(SERVER_PORT);

    if (connect(client_socket, (struct sockaddr *)&server_address, sizeof(server_address)) < 0)
    {
        perror("Error connecting to the university server");
        exit(EXIT_FAILURE);
    }

    char request_type[] = "RESERVE_EXAM";
    printf("%s", course);
    FullWrite(client_socket, request_type, sizeof(request_type));
    sleep(3);
    FullWrite(client_socket, course, strlen(course));
    sleep(3);
    FullWrite(client_socket, date, strlen(date));

    // Receive and forward the reservation confirmation to the student client
    char confirmation[100];
    int bRead = read(client_socket, confirmation, sizeof(confirmation));
    confirmation[bRead] = '\0';
    FullWrite(student_socket, confirmation, strlen(confirmation));

    close(client_socket);
}
```

2.2.3 Richiesta di Date degli Esami

La funzione request_exam_dates invia richieste al Server Universitario e inoltra le date degli esami al client studente. Questa funzione gestisce la comunicazione tra la Segreteria e gli studenti.

```c
void request_exam_dates(SOCKET student_socket, const char *course)
{
    // Request exam dates from the university server
    SOCKET client_socket = socket(AF_INET, SOCK_STREAM, 0);

    struct sockaddr_in server_address;
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = inet_addr(SERVER_IP);
    server_address.sin_port = htons(SERVER_PORT);

    if (connect(client_socket, (struct sockaddr *)&server_address,
    sizeof(server_address)) < 0)
    {
        perror("Error connecting to the university server");
        exit(EXIT_FAILURE);
    }

    char request_type[] = "REQUEST_EXAM_DATES";
    FullWrite(client_socket, request_type, sizeof(request_type));
    printf("Request forwarded to the server");
    sleep(3);
    FullWrite(client_socket, course, strlen(course));
    printf("Course forwarded to the server\n");

    // Receive available exam dates from the university server
    char exam_dates[500];
    int nread = read(client_socket, exam_dates, sizeof(exam_dates))
    ;
    exam_dates[nread] = '\0';

    // Forward the dates to the student client
    FullWrite(student_socket, exam_dates, strlen(exam_dates));

    close(client_socket);
}
```

2.3 Studente

Lo studente ha accesso al sistema universitario attraverso un'interfaccia intuitiva dove può richiedere la disponibilità degli esami ed effettuare una prenotazione ad un esame. Nelle sezioni seguenti, esploreremo dettagliatamente le principali funzioni messe a disposizione dello studente, comprendendo il processo di connessione, la richiesta di disponibilità degli esami, la prenotazione degli esami e l'opzione di uscita.

2.3.1 Avvio Connession

La funzione establish_connection gestisce il processo di connessione del cliente al server universitario.

```c
void establish_connection(SOCKET *client_socket) {
        *client_socket = socket(AF_INET, SOCK_STREAM, 0);
        struct sockaddr_in server_address = {0};
```

```
4           server_address.sin_family = AF_INET;
5           server_address.sin_addr.s_addr = inet_addr(SERVER_IP);
6           server_address.sin_port = htons(PORT);
7
8           if (connect(*client_socket, (struct sockaddr*)&
    server_address, sizeof(server_address)) == -1) {
9               printf("Connection failed\n");
10              close(*client_socket);
11          } else {
12              printf("Connection established\n");
13          }
14  }
```

2.3.2 Richiesta disponibilità di un esame

Questa funzione è dedicata alla richiesta della disponibilità degli esami. Dopo aver stabilito una connessione con il server, l'utente può inserire il nome dell'esame di interesse, ricevendo quindi una risposta che elenca le date disponibili. Nel caso in cui non ci siano date disponibili, il sistema informa l'utente di tale situazione.

```
1   void request_exam_availability(char course[]) {
2       SOCKET client_socket;
3       int byte_read = 0;
4
5       establish_connection(&client_socket);
6
7       char request_type[] = "REQUEST_EXAM_DATES";
8       FullWrite(client_socket, request_type, sizeof(request_type));
9       printf("Sending request: %s\n", request_type);
10      sleep(3);
11      FullWrite(client_socket, course, strlen(course));
12
13      char exam_dates[500];
14      byte_read = read(client_socket, exam_dates, sizeof(exam_dates))
        ;
15      if (byte_read == 0)
16          printf("** No available exam dates for %s **\n", course);
17      else if (byte_read > 0) {
18          exam_dates[byte_read] = '\0';
19          printf("\nAvailable exam dates for %s:\n%s", course,
    exam_dates);
20      } else
21          perror("Read error");
22
23      close(client_socket);
24  }
```

2.3.3 Prenotazione ad un esame

La prenotazione di un esame è semplificata attraverso la funzione reserve_exam. Dopo aver inserito il nome dell'esame e la data desiderata, il sistema comunica con il server per confermare la prenotazione. L'utente riceverà un messaggio di conferma che attesta la corretta registrazione della prenotazione.

```
void reserve_exam(const char* course, const char* date) {
    SOCKET client_socket;
    int byte_read = 0;

    establish_connection(&client_socket);

    char request_type[] = "RESERVE_EXAM";
    FullWrite(client_socket, request_type, sizeof(request_type));
    sleep(3);
    FullWrite(client_socket, course, strlen(course));
    sleep(3);
    FullWrite(client_socket, date, strlen(date));

    char confirmation[100];
    byte_read = read(client_socket, confirmation, sizeof(
    confirmation));
    confirmation[byte_read] = '\0';
    printf("\n Reservation confirmation: %s ", confirmation);

    close(client_socket);
}
```

Capitolo 3

Funzionamento

Esaminiamo attentamente il comportamento delle tre classi nelle specifiche situazioni stabilite, seguendo l'avvio dei terminali nell'ordine precedentemente indicato.

3.1 Server Universitario

Il server universitario ha il compito fondamentale di soddisfare tutte le richieste provenienti dagli studenti e dalla segreteria. Una volta stabilita una connessione, il server comunicherà quando è pronto a ricevere richieste. In caso di errori durante il processo, fornirà opportune segnalazioni.

Nel caso in cui uno studente richieda la disponibilità delle date d'esame, il server si comporterà nel seguente modo: il server risponde fornendo le informazioni sulle date disponibili. Questo processo implica la lettura dei dati dal file "exams.txt", dove sono archiviate le informazioni sugli esami, e la successiva trasmissione delle date disponibili allo studente. Tale approccio consente una risposta tempestiva e precisa alle richieste degli studenti, garantendo che abbiano accesso alle informazioni aggiornate in modo efficiente e accurato.

Nel caso in cui uno studente desideri prenotarsi per un esame: il server assume il compito di registrare le informazioni relative alla prenotazione nel file ***"reservations.txt"***. Questo il corso per cui si sta prenotando e il numero di prenotazione associato. Questo processo garantisce che le prenotazioni degli studenti siano correttamente archiviate per una gestione efficiente e trasparente.

Se invece la segreteria intende aggiungere un nuovo esame: il server assume la responsabilità di registrare le informazioni relative al nuovo esame nel file ***"exams.txt"***, se ciò è possibile. Successivamente, il server invierà una notifica alla segreteria contenente le informazioni essenziali relative all'aggiunta del nuovo esame. Questo processo assicura che le informazioni siano adeguatamente archiviate e che la segreteria sia tempestivamente informata dei cambiamenti nel sistema universitario.

3.2 Segreteria

La segreteria deve essere in grado di mettersi in ascolto delle richieste degli studenti e può anche inviare richieste al server per aggiungere un nuovo esame. Abbiamo messo a disposizione un menu che riporta queste opzioni.

Vediamo nel dettaglio come si comporta in queste casistiche:

1. Aggiungere un nuovo esame: Per aggiungere un nuovo esame, il sistema richiede innanzitutto l'inserimento del nome dell'esame. Successivamente, l'utente deve specificare la data in cui desidera inserire l'esame. Una volta completata questa operazione, il sistema invierà un messaggio di conferma contenente i dati appena inseriti.

2. Ascoltare le richieste degli studenti: La segreteria è in grado di ascoltare e gestire le richieste degli studenti. Questa funzionalità consente uno scambio bidirezionale di informazioni tra gli studenti e la segreteria. Gli studenti possono fare domande ed essa si interfaccerà con il server universitario, ottenere informazioni o effettuare prenotazioni, e la segreteria è pronta a rispondere e assistere nelle diverse esigenze degli studenti.

3.3 Studente

Lo studente ha la possibilità di eseguire diverse funzionalità, come la **richiesta della disponibilità dell'esame** e **la prenotazione per un esame**. Vedremo nel dettaglio entrambe le opzioni. Inoltre, ha la possibilità di decidere di uscire dal programma.

Dopo l'avvio, lo studente può selezionare una delle tre opzioni disponibili:

1. Richiesta della disponibilità dell'esame: Lo studente può selezionare l'opzione per richiedere la disponibilità delle date d'esame. In seguito alla scelta, il server universitario risponderà alla segreteria che le metterà a disposizione dello studente le informazioni sulle date disponibili per l'esame richiesto.

2. Prenotazione per un esame: Lo studente può scegliere di prenotarsi per un esame selezionando l'apposita opzione. Successivamente, inserirà il nome dell'esame e la data desiderata in formato DD/MM/YY. Il server universitario gestirà la prenotazione, registrando le informazioni nel file "reservations.txt" e assegnando un numero di prenotazione. Lo studente riceverà quindi un messaggio di conferma con il numero di prenotazione.

3. Exit: Lo studente ha la possibilità di uscire dal sistema selezionando l'opzione "Exit". In questo caso, il programma si chiuderà e lo studente verrà disconnesso dal sistema universitario.