Università degli Studi di Napoli "Parthenope"
Facoltà di Scienze e Tecnologie
Corso di laurea in Informatica

Documentazione Reti di Calcolatori

Traccia: Università

Docente
Emanuel Di Nardo

Candidato
Di Palo Lorenzo
Matricola: 0124002580

Anno Accademico 2023-2024

Indice

Capitolo 1

Introduzione

1.1 Obiettivo

Realizzare un'applicazione client/server parallelo per gestire gli esami universitari

Studente

- 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

Descrizione dell'architettura

2.1 Architettura

Il sistema è costituito da 2 architetture client/server (I/O Multiplexing)

- **Server/Segreteria:** La prima connessione che avviene è questa.
  Quando viene eseguito lo script `start.sh`
  SERVER - SEGRETERIA - STUDENTE (2.1.1) vengono avviati
  il *Server* attende che la *Segreteria* si connetta.

- **Segreteria/Studente:** La seconda connessione avviene dopo che la prima non ha riportato errori.
  Se quest'ultima non riporta errori,
  Segreteria attende l'inserimento delle Credenziali da parte dello *Studente*.

2.1.1 Componenti dell'architettura

- **Studente:** Fornisce interfaccia a linea di comando per:

  - **Visualizza Appelli:**
    Permette allo *Studente* di scegliere tra la visualizzazione di tutti gli appelli nel suo corso o di un appello specifico.
  - **Prenota Appello:**
    Richiede all'utente il nome del codice dell'esame al quale vuole prenotarsi e lo prenota nel primo appello disponibile.
  - **Logout:**
    elimina le informazioni della sessione

- **Segreteria:** Fornisce interfaccia a linea di comando per:

  - **Gestire Studenti:**
    Gestisce le Richieste da parte di *Studente*
  - **Inserire Appelli:** Inserisce nuovi appelli per esami esistenti in *Server*

- **Server:** Fornisce a *Segreteria* la possibilità di:

  - **Aggiunta Appello:**

  - **Aggiunta Prenotazione:**

2.1.2 Flusso di Comunicazione

Server: Attende connessioni.
Segreteria: Dopo aver stabilito una connessione con *Server*, *Segreteria* è pronta a ricevere nuove connessioni.
Studente: Dopo aver stabilito una connessione, *Studente* invia le credenziali a *Segreteria* per l'autenticazione.
Una volta autenticato, *Studente* può eseguire le Operazioni descritte in 2.1.1.

Capitolo 3

Dettagli implementativi dei client/server

3.1 Studente

3.1.1 Connessione alla Segreteria

Studente stabilisce una connessione con *Segreteria* utilizzando una socket, specificandone
Dominio(AF_INET)
Tipo ($SOCK\_STREAM$)
Protocollo (0)

```
1
2    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
3        perror("Errore nella creazione della socket!");
4        exit(1);
5    }
6    servaddr.sin_family = AF_INET;
7    if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <=
         0) {
8        fprintf(stderr, "Errore inet_pton per %s\n", argv[1])
             ;
9        exit(1);
10   }
11   servaddr.sin_port = htons(PORT_CLIENT);
12
13   if (connect(sockfd, (struct sockaddr *) &servaddr,
         sizeof(servaddr)) < 0) {
14       perror("Errore nella connect: ");
15       printf("\n");
16       exit(1);
17   }
```

Listing 3.1: Codice Connessione Socket

3.1.2 Autenticazione

Studente inserisce credenziali e se non sono presenti nel database viene terminata l'esecuzione.

```
1    printf("LOGIN\n");
2    printf("Inserire matricola: ");
3    scanf("%d", &mat);
4
5    /**
6    * Pulisco il buffer di input.
7    */
8    while ((c = getchar()) != '\n' && c != EOF);
9
10   printf("Inserire password: ");
11   fgets(pass, sizeof(pass), stdin);
12   pass[strlen(pass) - 1] = 0;
13
14   write(sockfd, &mat, sizeof(mat));
15   write(sockfd, pass, sizeof(pass));
16
17   char state[255] = {0};
18   read(sockfd, state, sizeof(state));
19
20   printf("\nEsito login: %s", state);
21   printf("\n");
22
23   if (strcmp(state, "credenziali non corrette, accesso
        negato!") == 0) {
24       exit(1);
25   }
```

Listing 3.2: Codice Autenticazione Segreteria

3.1.3 Scelta Operazione

Studente può effettuare una delle scelte riportate in 2.1.1

```
1    printf("\nInserire il numero relativo all'operazione che
         si vuole effettuare:\n");
2    printf("1 - Visualizza appelli disponibili\n");
3    printf("2 - Prenota un appello\n");
4    printf("3 - Logout\n");
5    printf("Scelta: ");
6    scanf("%d", &request);
7    printf("\n");
8    /*** pulisce buffer ***/
9    write(sockfd, &request, sizeof(request));
```

Listing 3.3: Codice Scelta Operazione

3.1.4 Ottieni Risposta

In base alla scelta effettuata, si ottiene una risposta dalla *Segreteria*. (Tentativo Formattazione Testo in C vano)

```
1   read(sockfd, &num_rows, sizeof(num_rows));
2
3   if (num_rows == 0) {
4       printf("\nNon esistono appelli disponibili!\n");
5   } else {
6       printf("\nAppelli disponibili:\n\n");
7
8       printf("| ID\tNome esame\tData | \n\n ");
9       for (int i = 0; i < num_rows; i++) {
10          read(sockfd, &id, sizeof(id));
11          read(sockfd, name, sizeof(name));
12          read(sockfd, date, sizeof(date));
13          printf("%d\t%s\t%s\n", id, name, date);
14          printf("-------------------------------\n");
15      }
16  }
```

Listing 3.4: Codice Ottieni Risposta (Richiesta Appelli)

3.2 Segreteria

3.2.1 Connessione al Server Universitario

La connessione al server universitario rispecchia la connessione avvenuta in 3.1

3.2.2 Hosting

La connessione da parte di *Studente* richiede una socket che rimanga in ascolto in attesa di nuove connessioni
In questo caso la socket è "listenfd"

```
1   listenfd = getListenSocket(AF_INET, SOCK_STREAM, 0);
2   secaddr.sin_family = AF_INET;
3   secaddr.sin_addr.s_addr = htonl(INADDR_ANY);
4   secaddr.sin_port = htons(PORT_SERVER);
5   bindListener(listenfd, secaddr, sizeof(secaddr));
6   /**
7   * Mettiamo il server in ascolto, specificando quante
        connessioni possono essere in attesa venire accettate
8   * tramite il secondo argomento della chiamata.
9   */
10  if ((listen(listenfd, 5)) < 0) {
11     perror("Errore nell'operazione di listen!");
12     exit(1);
13  }
```

Listing 3.5: Codice Ascolto Socket

3.2.3 Connessione al DB "UNIVERSITA"

Se la connessione va a buon fine, *conn* deterrà l'entry point per effettuare query sul DB

```
1    conn = mysql_init(NULL);
2    if (conn == NULL) {
3       fprintf(stderr, "mysql_init() fallita\n");
4       exit(1);
5    }
6    if (mysql_real_connect(conn, "localhost", "nonnt66", "
       password", "universita", 3306, NULL, 0) == NULL) {
7       fprintf(stderr, "mysql_real_connect() fallita: %s\n",
            mysql_error(conn));
8       mysql_close(conn);
9       exit(1);
10   } else {
11      printf("Connessione al database avvenuta con successo
          \n");
12   }
```

Listing 3.6: Codice Connessione DB

3.2.4 Descrittori

- Dichiarazione Descrittori:

  Identifico 3 set di descrittori:

  - **master_set:** l'insieme di descrittori che verrà passato alla funzione select, settato inizialmente a zero.
  - **read_set:** l'insieme di descrittori che verifica quale descrittore è pronto in lettura
  - **write_set:** l'insieme di descrittori che verifica quale descrittore è pronto in scrittura

Inizialmente i vene impostato il massimo descrittore, scegliendo, tra la socket dello *Studente* e la socket del *Server*

```
1   fd_set read_set, write_set, master_set;
2   int max_fd;
3   FD_ZERO(&master_set);
4
5   FD_SET(sockfd, &master_set);
6   max_fd = sockfd;
7
8   FD_SET(listenfd, &master_set);
9   max_fd = max(max_fd, listenfd);
```

Listing 3.7: Codice Descrittori

- Selezione Descrittori: Si impostano i descrittori al valore della master_set e si esegue la select per vedere se ci sono connessioni pronte

```
1   if (select(max_fd + 1, &read_set, &write_set, NULL,
        NULL) < 0) {
2       perror("Errore nell'operazione di select!");
3   }
```

Listing 3.8: Codice Descrittori Select

- Verifica Descrittore Pronto: Vado a verificare quale è il descrittore pronto in lettura/scrittura rispettivamente lato *Studente*, *Server*

```
1   if (FD_ISSET(listenfd, &read_set)) {
2    /**
3    * La system call accept permette di accettare una
         nuova connessione (lato server) in entrata da un
         client.
4    */
5    if ((client_sockets[dim].connfd = accept(listenfd, (
         struct sockaddr *) NULL, NULL)) < 0) {
6       perror("Errore nell'operazione di accept!");
7    } else {
8       /**
9       * Si aggiunge il descrittore legato alla nuova
            connessione da uno studente all'interno dell'
            array di
10      * descrittori master_set e si ricalcola il numero
            di posizioni da controllare nella select.
11      */
12      FD_SET(client_sockets[dim].connfd, &master_set);
13      max_fd = max(max_fd, client_sockets[dim].connfd);
14      /*** esegue oprazioni SQL ***/
15      /*** Itera sui client connessi ***/
16      if (FD_ISSET(client_sockets[i].connfd, &read_set)
            && client_sockets[i].connfd != -1) {
17         read(client_sockets[i].connfd, &behaviour,
              sizeof(behaviour));
```

Listing 3.9: Codice Accettazione Studente - Lettura Scelta Studente

3.3 Server

3.3.1 Operazioni Server

- Inizializza una socket **listenfd** 3.5
- Dichiara Insieme di **Descrittori** 3.7
- Seleziona **Descrittore** 3.8
- Verifica ed Utilizza **Descrittore** 3.9
- Aggiunge Prenotazione

```
1   read(connfd, &id, sizeof(id));
2   read(connfd, &mat, sizeof(mat));
3
4   char query_verifica_prenotazione[500];
5
6   snprintf(query_verifica_prenotazione, sizeof(
       query_verifica_prenotazione),
7   "select a.id_esame, MIN(data_appello) "
8   "from appello a "
9   "join esame e "
10  "join studente s "
11  "where a.id_esame = %d "
12  "and data_appello > sysdate() "
13  "and anno_corso_studente >= e.anno_corso_esame "
14  "and s.mat_studente = '%d' "
15  "and (select count(*) from supera s where s.
       mat_studente = '%d' and s.id_esame = %d) = 0 "
16  "group by a.id_esame;", id, mat, id, mat);
17
18  if (mysql_query(conn, query_verifica_prenotazione)
       != 0) {
19     fprintf(stderr, "\nmysql_query(
          query_verifica_prenotazione): %s\n",
          mysql_error(conn));
20     write(connfd, mysql_error(conn), strlen(
          mysql_error(conn)));
21  }
22
23  MYSQL_RES *res_qvp = mysql_store_result(conn);
24  if (res_qvp == NULL) {
```

Listing 3.10: Codice Query Aggiungi Prenotazione

```
1       fprintf(stderr, "\nmysql_store_result(
           query_verifica_prenotazione): %s\n",
           mysql_error(conn));
2       write(connfd, mysql_error(conn), strlen(
           mysql_error(conn)));
3       }
4
5       unsigned int rows = mysql_num_rows(res_qvp);
6       /***
7       * Se non esiste un appello con questo id e questa
            data allora restituisco un errore
8       */
9       if (!rows) {
10      const char *err = "invalid id or data!";
11      write(connfd, err, strlen(err));
12      } else {
13      MYSQL_ROW row_qvp = mysql_fetch_row(res_qvp);
14      int id_esame = strtol(row_qvp[0], NULL, 10);
15      char data_appello[12];
16      strcpy(data_appello, row_qvp[1]);
17
18      char inserimento[255];
19      snprintf(inserimento, sizeof(inserimento),
20      "insert into prenota (mat_studente, id_esame,
           data_prenotazone) VALUES ('%d', %d, sysdate())
           ;",
21      mat, id_esame);
22      if (mysql_query(conn, inserimento) != 0) {
23          write(connfd, mysql_error(conn), strlen(
               mysql_error(conn)));
24      } else {
25          char res[255];
26          snprintf(res, sizeof(res),
27          printf("ttok");
28          write(connfd, res, strlen(res));
29      }
30      }
```

Listing 3.11: Codice Query Aggiungi Prenotazione

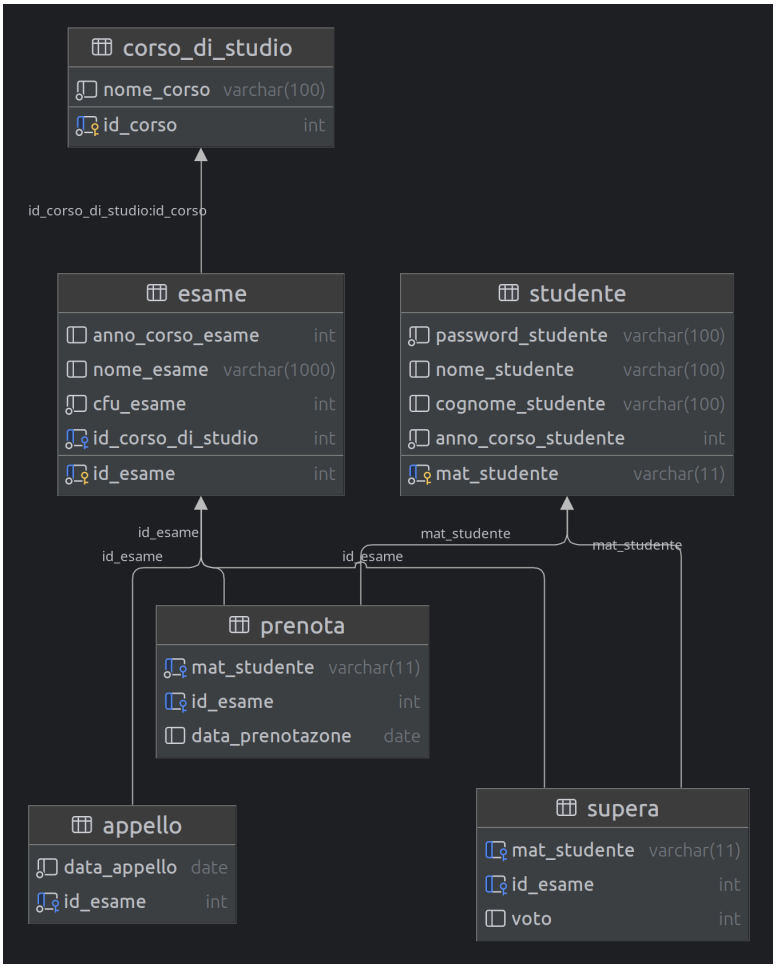

Figura 3.1: Schema Database Univerisità

- **Schema Database**