



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

1	Introduzione	1
1.1	Obiettivo	1
2	Descrizione dell'architettura	2
2.1	Architettura	3
2.1.1	Componenti dell'architettura	4
2.1.2	Flusso di Comunicazione	4
3	Dettagli implementativi dei client/server	5
3.1	Studente	6
3.1.1	Connessione alla Segreteria	6
3.1.2	Autenticazione	7
3.1.3	Scelta Operazione	8
3.1.4	Ottieni Risposta	9
3.2	Segreteria	10
3.2.1	Connessione al Server Universitario	10
3.2.2	Hosting	10
3.2.3	Connessione al DB "UNIVERSITA"	11
3.2.4	Descrittori	12
3.3	Server	14
3.3.1	Operazioni Server	14

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

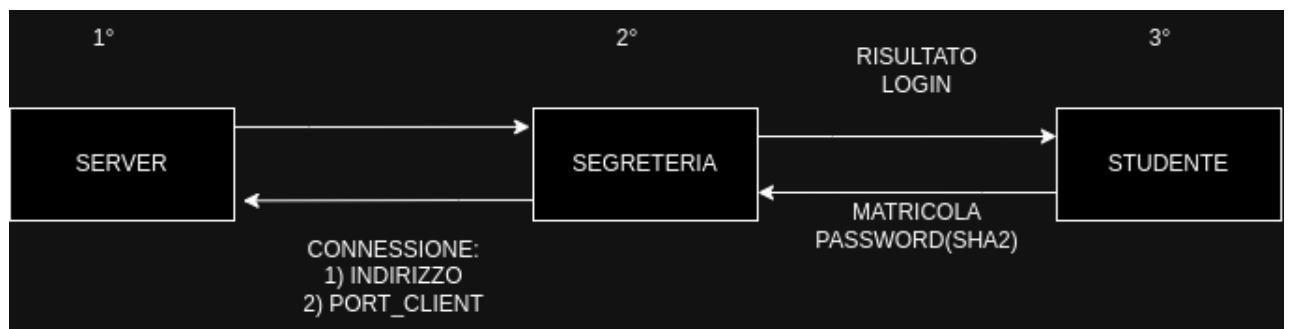
Capitolo 2

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) <=
8         0) {
9         fprintf(stderr, "Errore inet_pton per %s\n", argv[1])
10        ;
11        exit(1);
12    }
13    servaddr.sin_port = htons(PORT_CLIENT);
14
15    if (connect(sockfd, (struct sockaddr *) &servaddr,
16        sizeof(servaddr)) < 0) {
17        perror("Errore nella connect: ");
18        printf("\n");
19        exit(1);
20    }
```

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
24     negato!") == 0) {
25     exit(1);
26 }
```

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 *Studiante* 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
8  * connessioni possono essere in attesa venire accettate
9  * tramite il secondo argomento della chiamata.
10 */
11 if ((listen(listenfd, 5)) < 0) {
12     perror("Errore nell'operazione di listen!");
13     exit(1);
14 }
```

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", "
7      password", "universita", 3306, NULL, 0) == NULL) {
8      fprintf(stderr, "mysql_real_connect() fallita: %s\n",
9          mysql_error(conn));
10     mysql_close(conn);
11     exit(1);
12 } else {
13     printf("Connessione al database avvenuta con successo
14         \n");
15 }
```

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 viene 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,
2      NULL) < 0) {
3      perror("Errore nell'operazione di select!");
  }

```

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
4       * nuova connessione (lato server) in entrata da un
5       * client.
6       */
7      if ((client_sockets[dim].connfd = accept(listenfd, (
8          struct sockaddr *) NULL, NULL)) < 0) {
9          perror("Errore nell'operazione di accept!");
10     } else {
11         /**
12          * Si aggiunge il descrittore legato alla nuova
13          * connessione da uno studente all'interno dell'
14          * array di
15          * descrittori master_set e si ricalcola il numero
16          * di posizioni da controllare nella select.
17          */
18         FD_SET(client_sockets[dim].connfd, &master_set);
19         max_fd = max(max_fd, client_sockets[dim].connfd);
20         /*** esegue operazioni SQL ***/
21         /*** Itera sui client connessi ***/
22         if (FD_ISSET(client_sockets[i].connfd, &read_set)
23             && client_sockets[i].connfd != -1) {
24             read(client_sockets[i].connfd, &behaviour,
25                 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(
7      query_verifica_prenotazione),
8      "select a.id_esame, MIN(data_appello) "
9      "from appello a "
10     "join esame e "
11     "join studente s "
12     "where a.id_esame = %d "
13     "and data_appello > sysdate() "
14     "and anno_corso_studente >= e.anno_corso_esame "
15     "and s.mat_studente = '%d' "
16     "and (select count(*) from supera s where s.
17         mat_studente = '%d' and s.id_esame = %d) = 0 "
18     "group by a.id_esame;", id, mat, id, mat);
19
20 if (mysql_query(conn, query_verifica_prenotazione)
21     != 0) {
22     fprintf(stderr, "\nmysql_query(
23         query_verifica_prenotazione): %s\n",
24         mysql_error(conn));
25     write(connfd, mysql_error(conn), strlen(
26         mysql_error(conn)));
27 }
28
29 MYSQL_RES *res_qvp = mysql_store_result(conn);
30 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_prenotazione) 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

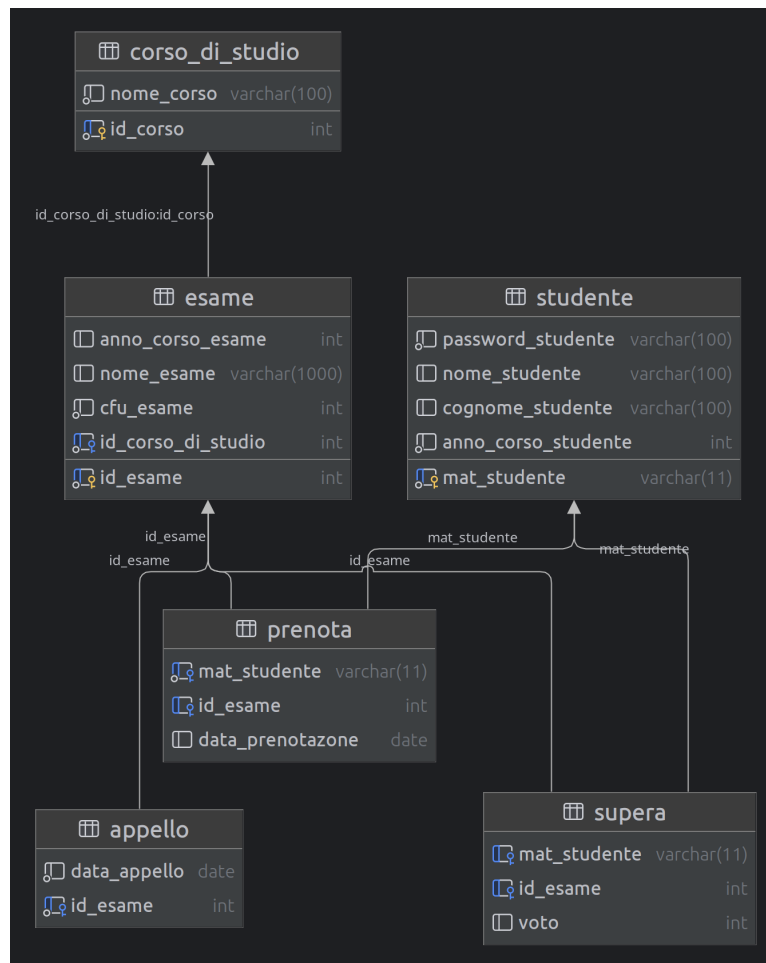


Figura 3.1: Schema Database Univerisità

- Schema Database