Università degli studi di Napoli "PARTHENOPE"

Reti di Calcolatori

Traccia - Università

Anno 2024/2025

Francesco Porritiello 0124002486
Francesco Esposito 0124002629

Contents

1 Introduzione

1.1 Obiettivo

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

- **Studente**
  1. Chiede alla segreteria se ci siano esami disponibili per un corso.
  2. Invia una richiesta di prenotazione di un esame alla segreteria.
- **Segreteria**
  1. Inserisce gli esami sul server dell'università (salvare in un file o conservare in memoria il dato).
  2. Inoltra la richiesta di prenotazione degli studenti al server universitario.
  3. Fornisce allo studente le date degli esami disponibili per l'esame scelto dallo studente.
- **Università**
  1. Riceve l'aggiunta di nuovi esami.
  2. Riceve la prenotazione di un esame.
  3. Ad ogni richiesta di prenotazione invia alla segreteria il numero di prenotazione progressivo assegnato allo studente e la segreteria a sua volta lo inoltra allo studente.

2 Descrizione

Il sistema è costituito da tre entità, Studente, Segreteria ed Università.

Essi fanno uso dell'architettura client/server sfruttando il protocollo TCP/IP per la comunicazione.

2.1 Architettura

Server e Segreteria fungono entrambi sia da client che da server, mentre lo Studente soltanto da client.

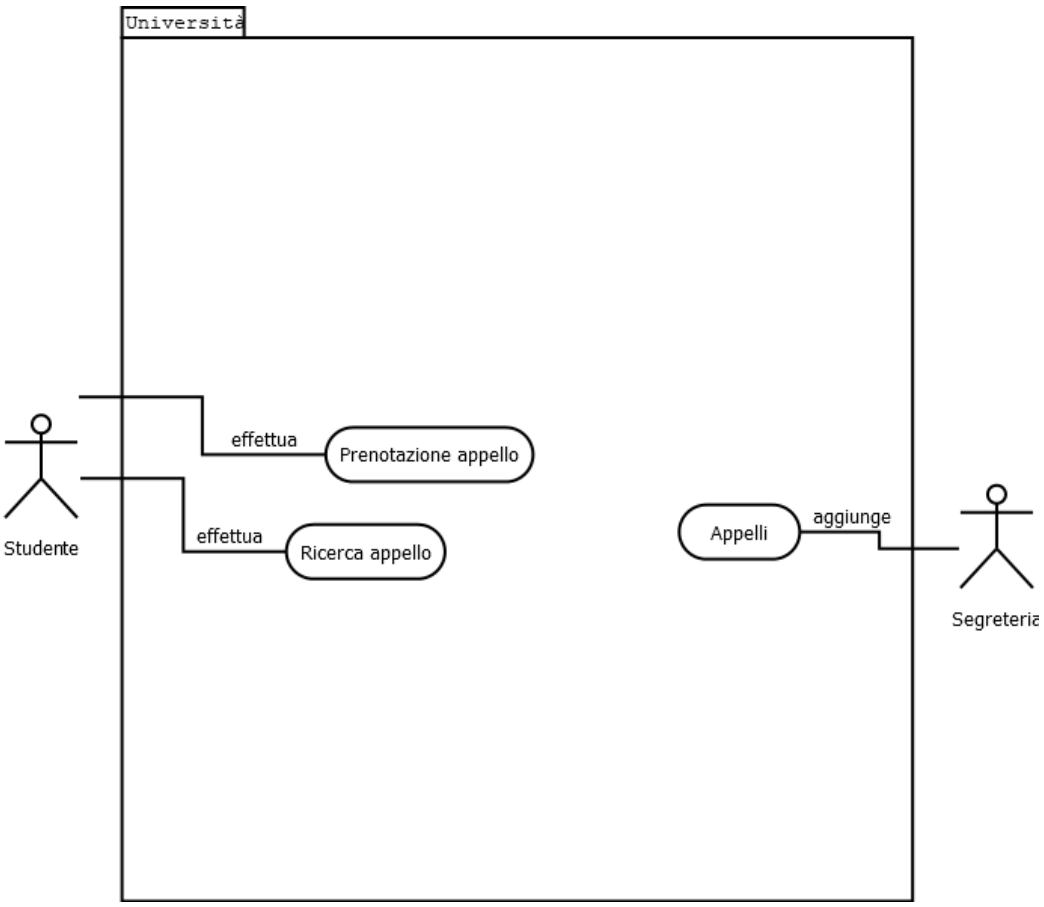
L'Università comunica in modo diretto esclusivamente con la Segreteria; quest'ultima viene usata come intermediara dallo Studente per effettuare richieste all'Università.

La sequenza con cui il sistema deve essere avviato è la seguente:

1. Server
2. Segreteria
3. Studente

Il Server Universitario, una volta avviato, attende che la Segreteria si connetta. Stabilita la connessione, la Segreteria, può adesso effettuare richieste all'Università, restando allo stesso tempo in attesa di connessioni da parte del client Studente per servire le sue richieste.

3 Use-Case


Universita
effettua
Prenotazione appello
effettua
Ricerca appello
Studente
Appelli
aggiunge
Segreteria

4 Componenti del Sistema

- Studente - Fornisce un'interfaccia a linea di comando per:
  1. Ricerca Appelli.
  2. Prenotazione Appelli.

- Segreteria - Si occupa di:
  1. Aggiungere Appelli.
  2. Gestire le richieste dello Studente mediando con l'Università quando effettua operazioni di ricerca e/o prenotazione appelli.

- Università - Si occupa di:
  1. Fornire le date degli Appelli disponibili.
  2. Salvare in un file gli Appelli aggiunti.
  3. Salvare in un file le prenotazioni agli Appelli.
  4. Inoltrare il numero di prenotazione progressivo ad uno Studente che ha effettuato una prenotazione ad un Appello.

5 Dettagli implementativi

Di seguito verranno descritti i dettagli implementativi del sistema:

- Università
- Segreteria
- Studente

5.1 Università

```
void gestisciRichiesta(int client_socket) {
    char buffer[BUFFER_SIZE] = {0};        // Lettura dei
       dati dal client
    char tipo_richiesta;                   // Tipi di
       richieste ('A', 'P', 'I')
    char nome_esame[BUFFER_SIZE] = {0};    // Nome esame
    char data_esame[BUFFER_SIZE] = {0};    // Data esame
    char linea[BUFFER_SIZE] = {0};         // Buffer per
       leggere la linea del file
    int esame_trovato = 0;                 // Flag per
       verificare se un esame e' trovato

    // Leggi la richiesta dal client
    read(client_socket, buffer, BUFFER_SIZE);
    tipo_richiesta = buffer[0]; // La prima lettera indica il
        tipo di richiesta

    if (tipo_richiesta == 'A') { // Aggiunta di un esame
        // Prendi tutto il contenuto fino alla newline
        sscanf(buffer + 1, "%[^\n]", buffer);

    // Trova la posizione dell'ultimo spazio, che separa il
       nome dell'esame dalla data
    char *space_pos = strrchr(buffer, ' ');

    if (space_pos != NULL) {
        // Calcola la lunghezza del nome dell'esame e copia
           il nome
        int nome_len = space_pos - buffer;
        strncpy(nome_esame, buffer, nome_len);
        nome_esame[nome_len] = '\0'; // Aggiunge il
           terminatore
```

```
25
26          // Copia la data dell'esame
27          strcpy(data_esame, space_pos + 1);
28      }
29
30          // Apri il file degli esami sia per la lettura che
                per la scrittura
31          FILE *file = fopen(EXAMS_FILE, "r+");
32          if (file == NULL) {
33              perror("Errore nell'apertura del file esami");
34              exit(EXIT_FAILURE);
35          }
36
37          // Crea un file temporaneo per scrivere i dati
                aggiornati
38          FILE *temp_file = fopen("temp.txt", "w");
39          if (temp_file == NULL) {
40              perror("Errore nell'apertura del file temporaneo"
                    );
41              fclose(file);
42              exit(EXIT_FAILURE);
43          }
44
45          // Leggi il file degli esami riga per riga
46          while (fgets(linea, sizeof(linea), file)) {
47              char esame[BUFFER_SIZE] = {0};
48              sscanf(linea, "%[^\n]", esame); // Prendi il nome
                    dell'esame
49
50              if (strstr(esame, nome_esame) == esame) {
51                  // Esame gia' esistente, aggiungi la nuova
                        data
52                  fprintf(temp_file, "%s %s\n", esame,
                        data_esame);
53                  esame_trovato = 1;
54              } else {
55                  // Copia la riga esistente nel file
                        temporaneo
56                  fprintf(temp_file, "%s", linea);
57              }
58          }
59
60          // Se l'esame non e' stato trovato, aggiungilo
61          if (!esame_trovato) {
```

```
62              fprintf(temp_file, "%s %s\n", nome_esame,
                    data_esame);
63          }
64
65          // Chiudi i file
66          fclose(file);
67          fclose(temp_file);
68          // Sostituisci il file originale con quello
                temporaneo
69          remove(EXAMS_FILE);
70          rename("temp.txt", EXAMS_FILE);
71
72          // Invia al client una conferma che l'esame e' stato
                aggiunto
73          send(client_socket, "Esame aggiunto con successo.\n",
                29, 0);
74      } else if (tipo_richiesta == 'P') { // Se la richieta e'
            quella di prenotazione
75
76          sscanf(buffer + 1, "%[^\n]", buffer);
77
78          // Apri il file delle prenotazioni
79          FILE *file = fopen(BOOKINGS_FILE, "a");
80          if (file == NULL) {
81              perror("Errore nell'apertura del file
                    prenotazioni");
82              exit(EXIT_FAILURE);
83          }
84
85          // Genera un numero di prenotazione casuale tra 1 e
                100
86          int numero_prenotazione = rand() % 100 + 1;
87
88          // Aggiungi la prenotazione al file
89          fprintf(file, "%s \n", buffer);
90          fclose(file);
91
92          // Invia la risposta al client
93          char risposta[BUFFER_SIZE] = {0};
94          snprintf(risposta, sizeof(risposta), "Prenotazione -
                %s. Numero prenotazione: %d\n", buffer,
                numero_prenotazione);
95          send(client_socket, risposta, strlen(risposta), 0);
96
```

```
97      } else if (tipo_richiesta == 'I') { // Richiesta per
            infromazioni su un esame
98          sscanf(buffer + 1, "%[^\n]", nome_esame);
99
100         // Apri il file degli esami in modalita' lettura
101         FILE *file = fopen(EXAMS_FILE, "r");
102         if (file == NULL) {
103             perror("Errore nell'apertura del file esami");
104             exit(EXIT_FAILURE);
105         }
106
107         // Cerca l'esame nel file
108         while (fgets(linea, sizeof(linea), file)) {
109             if (strstr(linea, nome_esame) != NULL) {
110                 send(client_socket, linea, strlen(linea), 0);
                        // Invia la riga al client
111                 esame_trovato = 1; // Segna che l'esame e'
                        stato trovato
112             }
113         }
114         fclose(file);
115         // Se l'esame non e' stato trovato, invia un
                messaggio al client
116         if (!esame_trovato) {
117             send(client_socket, "Esame non trovato.\n", 19,
                    0);
118         }
119     }
120 }
```

La funzione **gestisciRichiesta**, legge i dati dal client segreteria e determina il tipo di richiesta (**A**: aggiunta, **P**: prenotazione, **I**: informazioni).

A seconda del tipo di richiesta, esegue operazioni appropriate.

5.2 Segreteria

```
void* gestisciRichiesta(void* arg) { // Funzione che gestisce le richieste effettuate dallo Studente.
    int client_socket = *(int*)arg; // Assegnazione della socket client.
    free(arg);

    char buffer[BUFFER_SIZE] = {0}; // Buffer per l'inoltro delle richieste.
    char risposta[BUFFER_SIZE] = {0}; // Buffer che conterra' la risposta del server Universitario per lo Studente.
    int sock_universita; // Socket server universitario.
    struct sockaddr_in server_addr_universita; // Definisce una struttura sockaddr_in per memorizzare l'indirizzo del server dell'universita'.

    // Leggi la richiesta dallo studente.
    if (read(client_socket, buffer, BUFFER_SIZE) < 0) {
        perror("Errore nella lettura della richiesta");
        close(client_socket);
        pthread_exit(NULL);
    }

    // Creazione del socket per la connessione con il server universitario.
    if ((sock_universita = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Errore nella creazione del socket per il server universitario");
        close(client_socket);
        pthread_exit(NULL);
    }

    server_addr_universita.sin_family = AF_INET; // Imposta la famiglia di indirizzi su AF_INET, specificando che si utilizzera' il protocollo IPv4.
    server_addr_universita.sin_port = htons(SERVER_PORT); // Imposta il numero di porta del server, convertendolo in network byte order con htons.

    if (inet_pton(AF_INET, SERVER_IP, &server_addr_universita.sin_addr) <= 0) { // Converte l'indirizzo IP del server da stringa a formato binario e lo memorizza nella struttura sockaddr_in.
```

```
28          perror("Indirizzo non valido o non supportato");
29          close(client_socket);
30          pthread_exit(NULL);
31      }
32
33      // Connessione al server universitario.
34      if (connect(sock_universita, (struct sockaddr *)&
            server_addr_universita, sizeof(server_addr_universita)
            ) < 0) {
35          perror("Connessione al server universitario fallita")
                ;
36          close(client_socket);
37          pthread_exit(NULL);
38      }
39
40      // Invia la richiesta al server universitario.
41      if (send(sock_universita, buffer, strlen(buffer), 0) < 0)
            {
42          perror("Errore nell'invio della richiesta al server
                universitario");
43          close(sock_universita);
44          close(client_socket);
45          pthread_exit(NULL);
46      }
47
48      // Ricevi la risposta dal server universitario.
49      int len = read(sock_universita, risposta, BUFFER_SIZE);
50      if (len < 0) {
51          perror("Errore nella lettura della risposta dal
                server universitario");
52          risposta[0] = '\0'; // Risposta vuota in caso di
                errore.
53      } else {
54          risposta[len] = '\0';
55      }
56
57      // Invia la risposta del server universitario allo
            studente.
58      if (send(client_socket, risposta, strlen(risposta), 0) <
            0) {
59          perror("Errore nell'invio della risposta allo
                studente");
60      }
61
62      close(sock_universita); // Chiusura delle socket.
```

```
63      close(client_socket);
64
65      pthread_exit(NULL); // Termina il thread corrente e
            restituisce un valore di uscita NULL.
66  }
```

La funzione **gestisciRichiesta**, gestisce le richieste dello studente, per poi inviarle al server universitario. Si occupa poi di inoltrare la risposta ricevuta dal server universitario allo studente.

5.3 Studente

```
1   void inviaRichiestaAllaSegreteria(char tipoRichiesta, char *
        matricola, char *nomeEsame, char *dataEsame) { // Funzione
        per la gestione della richiesta degli appelli disponibili
        .
2       int sock_segreteria; // Socket della segreteria.
3       struct sockaddr_in server_addr_segreteria; // Indirizzo
            della Segreteria.
4       char buffer[BUFFER_SIZE] = {0};
5
6       // Creazione del socket per la connessione con la
            segreteria
7       if ((sock_segreteria = socket(AF_INET, SOCK_STREAM, 0)) <
            0) {
8           perror("Errore nella creazione del socket");
9           exit(EXIT_FAILURE);
10      }
11
12      server_addr_segreteria.sin_family = AF_INET; // Si
            specifica che l'indirizzo del server utilizzera' l'
            indirizzamento ipv4.
13      server_addr_segreteria.sin_port = htons(SECRETARY_PORT);
            // Converte il numero di porta della Segreteria.
14
15      if (inet_pton(AF_INET, SERVER_IP, &server_addr_segreteria
            .sin_addr) <= 0) { // Conversione dell'indirizzo IP
            del server da formato testuale a formato binario.
16          perror("Indirizzo non valido o non supportato");
17          exit(EXIT_FAILURE);
18      }
19
20      // Connessione alla segreteria
```

```
21      if (connect(sock_segreteria, (struct sockaddr *)&
            server_addr_segreteria, sizeof(server_addr_segreteria)
            ) < 0) {
22          perror("Connessione fallita");
23          exit(EXIT_FAILURE);
24      }
25
26      // Invia la richiesta alla segreteria.
27      if (tipoRichiesta == 'P') {
28          snprintf(buffer, sizeof(buffer), "%c%s %s %s",
                tipoRichiesta, matricola, nomeEsame, dataEsame);
                // Viene inserito nel buffer la tipologia della
                richiesta, Esame e Data esame.
29      } else {
30          snprintf(buffer, sizeof(buffer), "%c%s",
                tipoRichiesta, nomeEsame); // Viene inserito nel
                buffer la tipologia della richiesta ed il nome
                esame,
31      }
32
33      send(sock_segreteria, buffer, strlen(buffer), 0); //
            Invio dei dati contenuti in buffer alla Segreteria.
34
35      // Lo studente riceve risposta dell'opportuna richiesta
            fatta.
36      int len = read(sock_segreteria, buffer, BUFFER_SIZE);
37      if (len > 0) {
38          buffer[len] = '\0';
39          printf("%s\n", buffer);
40      }
41
42      close(sock_segreteria); //Chiusura della connessione con
            la Segreteria.
43  }
```

La funzione **inviaRichiestaAllaSegreteria**, si occupa dell'invio di richieste alla segreteria.

6 Compilazione ed Esecuzione

Di seguito verranno mostrati i comandi per compilare ed eseguire il sistema su piattaforme *Unix-Like.

- Università
    1. Compilazione: "gcc server.c -o server"
    2. Esecuzione: "./uni"

- Segreteria
    1. Compilazione: "gcc segre.c -o segre"
    2. Esecuzione: "./segre"

- Studente
    1. Compilazione: "gcc stu.c -o stu"
    2. Esecuzione: "./stu"

* Non possono essere compilati ed eseguiti su sistemi operativi Windows in quanto sono stati pensati per sfruttare librerie e system-call POSIX.