Programming Assignment

Studente: **Luca Stanzione** - Mat: **0124001873** - Codice Gruppo: **x6p4216hzy** 09/12/2024

Introduzione

Il seguente documento è redatto al fine di mostrare le specifiche tecniche, l'implementazione e l'esecuzione di un'applicazione client-server parallela. L'applicazione di cui sopra deve implementare un sistema di gestione di esami universitari nel quale partecipano tre figure principali: la **Segreteria**, lo **Studente** ed il **Server Universitario**.

Le funzionalità offerte sono le seguenti:

- **Segreteria**: (I) Inserisce gli esami sul server dell'università; (II) Inoltra la richiesta di prenotazione degli studenti al server universitario; (III) Fornisce allo studente le date disponibili per l'esame scelto dallo studente;
- **Studente**: (I) Chiede alla segreteria se ci siano esami disponibili per un corso (lista date); (II) Invia una richiesta di prenotazione di un esame alla segreteria;
- **Server Universitario**: (I) Riceve l'aggiunta di nuovi esami; (II) Riceve la prenotazione di un esame.

Design

L'applicativo sopra descritto può essere implementato come un sistema client-server nel quale ci sono tre applicativi principali, ognuno dei quali implementa — rispettivamente — le funzionalità di Segreteria, Studente e Server Universitario, rappresentata in Figure 1.

Figure 1: Rappresentazione alto livello del sistema.

Studente L'applicativo Studente interagisce come client e funziona nel seguente modo: all'avvio viene richiesto all'utente di inserire il codice di un esame di interesse (ad es. INFORMATICA01) e l'azione da compiere, ovvero 1. per richiedere la lista di date per l'esame, 2. per prenotarsi ad determinato esame. Una volta inseriti codice ed azione, è richiesto l'inserimento della matricola. Queste informazioni sono impacchettate ed inviate all'applicativo Segreteria, il quale fornisce una risposta adeguata al seguito dell'azione. Per semplicità, lo studente potrà prenotarsi solo al primo appello disponibile. Per conoscere la lista delle date di un esame e prenotarsi al primo appello disponibile, l'utente deve avviare l'applicativo Studente, inserire la matricola, inserire il

codice dell'esame e digitare 1. In seguito a questa azione verrà mostrata la lista di date per l'esame scelto e, per prenotarsi bisognerà inserire di nuovo il codice dell'esame e digitare 2. Il diagramma di flusso ed il diagramma di interazione , per una generica azione, tra lo Studente e la Segreteria sono mostrati, rispettivamente, in Figure 2 e Figure 3.

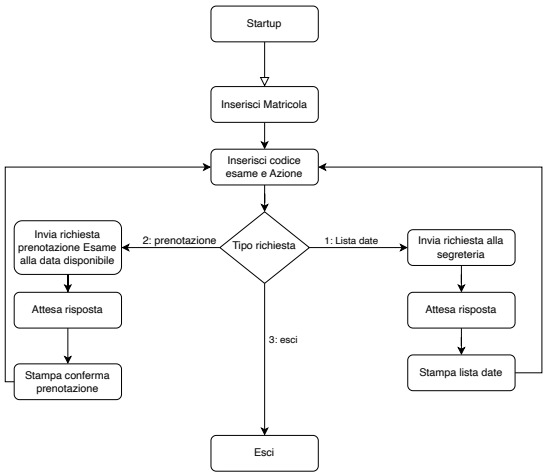

Figure 2: Diagramma di flusso dell'applicativo Studente.

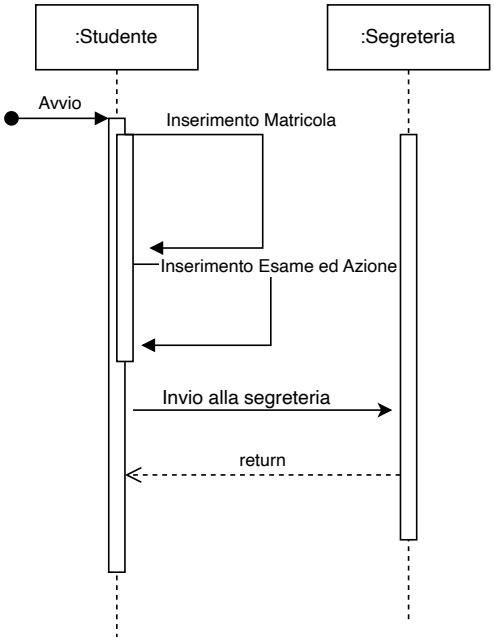

Figure 3: Rappresentazione alto livello del sistema.

Server Universitario Il server universitario funge da applicativo server per la segreteria. È sempre attivo e, appena interagisce con la segreteria riceve tutti gli esami disponibili con le loro date. Tiene in memoria queste informazioni insieme ad una lista di studenti prenotati agli appelli.

Fornisce la possibilità di memorizzare nuovi esami o di salvare nuove prenotazioni. Il sequence diagram relativo a questo applicativo verrà mostrato successivamente insieme al diagramma della Segreteria.

Segreteria L'applicativo della Segreteria, parte fondamentale dell'architettura, funge da server per lo Studente e da client per Server Universitario. Al suo avvio, fornisce la lista degli esami — pre-caricati in memoria — nel server universitario. Una volta fatta questa operazione, si mette in attesa di eventuali connessioni da parte dell'applicativo degli Studenti il quale può inviare contattare la segreteria, inviandogli matricola, azione (1. lista esami o 2. prenotazione al primo appello disponibile). Le operazioni eseguite, a seconda dell'azione, sono le seguenti:

- **1. Lista date**: cerca, all'interno della propria memoria, la lista delle date per il codice fornito dallo studente e, se trovato, restituisce tutti gli appelli per l'esame. Se non trovato, viene restituito un errore di ricerca.
- **2. Prenotazione Appello**: cerca, all'interno della propria memoria, la lista delle date per il codice fornito dallo studente e, se trovato, invia al Server Universitario la richiesta di prenotazione del primo appello disponibile da parte dello studente (utilizzando la matricola di quest'ultimo).

Come per Studente, il diagramma di flusso generico dell'applicazione, insieme all'interazione generica tra Studente, Segreteria e Server sono mostrate rispettivamente in Figure 4 e 5.

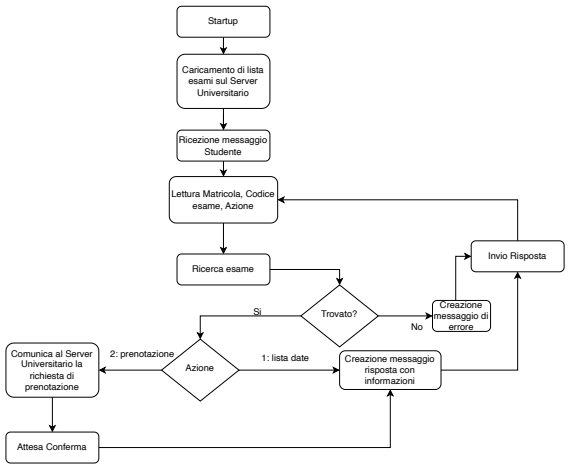

Figure 4: Diagramma di flusso dell'applicativo Segreteria.

Implementazione e codice rilevante

I tre applicativi sopra descritti sono stati implementati in linguaggio C, utilizzando le socket definite `sys/socket.h` per la comunicazione tra processi.

Sono state definite due funzioni di lettura e scrittura sulle socket chiamati `FullWrite` e `FullRead`, definite nel file header `fullops.h` e presentate nel listing 1.

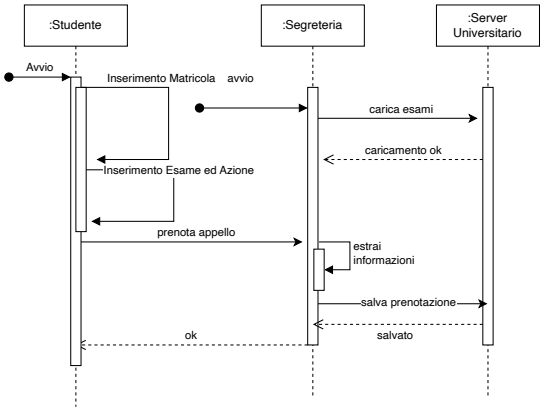

Figure 5: Sequence diagram generale.

Listing 1: FullRead e FullWrite.

```c
#include "fullops.h"
ssize_t FullRead(int fd, void *buf, size_t count)
{
    size_t nleft;
    ssize_t nread;
    nleft = count;
    while (nleft > 0) {             /* repeat until no left */
      if ( (nread = read(fd, buf, nleft)) < 0) {
          if (errno == EINTR) {   /* if interrupted by system call */
                continue;           /* repeat the loop */
            } else {
                exit(nread);      /* otherwise exit */
          }
        } else if (nread == 0) {    /* EOF */
            break;                  /* break loop here */
        }
        nleft -= nread;             /* set left to read */
        buf += nread;                /* set pointer */
    }
      buf = 0;
    return (nleft);
}

ssize_t FullWrite(int fd, void *buf, size_t count)
{
    size_t nleft;
    ssize_t nwritten;
    nleft = count;
    while (nleft > 0) {             /* repeat until no left */
      if ( (nwritten = write(fd, buf, nleft)) < 0) {
          if (errno == EINTR) {   /* if interrupted by system call */
              continue;           /* repeat the loop */
```

```
33            } else {
34                exit(nwritten);   /* otherwise exit with error */
35            }
36        }
37        nleft -= nwritten;          /* set left to write */
38        buf += nwritten;             /* set pointer */
39      }
40      return (nleft);
41  }
```

Un'altra parte fondamentale dell'implementazione riguarda la definizione di strutture dati atte a contenere informazioni sugli esami, sugli appelli e sui dati scambiati sulle socket. Tutto ciò è definito in un file header chiamato `common.h`, mostrato nel listing 2.

Listing 2: common.h

```
1  #ifndef COMMON_H_
2  #define COMMON_H_
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <stdbool.h>
8
9  const char azione1[] = {'L','I','S','T','A','D','A','T','E','\0'};
10 const char azione2[] = {'P','R','E','N','O','T','A','Z','I','O','N','E','\0'};
11 const int maxSize = 100;
12 const int segreteriaPort = 1024;
13 const int serverPort = 1026;
14 const int maxNumAppelli = 100;
15 const char *ip = "127.0.0.1";
16 typedef char Esame[maxSize];
17 typedef char Azione[maxSize];
18 typedef char Data[maxSize];
19 typedef char Matricola[10];
20
21 typedef struct Appello_str{
22   Esame esame;
23   Data data;
24 } Appello;
25
26 typedef struct RichiestaStudente_str{
27   Esame esame;
28   Azione azione;
```

```
29    Matricola matricola;
30  } RichiestaStudente;
31
32  typedef struct RispostaSegreteriaLista_str{
33      Appello appello;
34  } RispostaSegreteriaLista;
35
36  typedef struct RispostaSegreteriaPrenotazione_str{
37      Appello appello;
38  } RispostaSegreteriaPrenotazione;
39
40  typedef struct RispostaSegreteria_str{
41    Esame esame;
42    Azione azione;
43    bool errore;// = false;
44    RispostaSegreteriaLista lista;// = NULL;
45    RispostaSegreteriaPrenotazione prenotazione;//= NULL;
46  } RispostaSegreteria;
47
48  typedef struct Prenotazione_str{
49    Appello appello;
50    Matricola matricola;
51  } Prenotazione;
52
53  typedef struct RichiestaSegreteria_str{
54    int type; // 1 per caricamento esami, 2 per caricamento appello studente
55    int numAppelli;
56    Appello appelli[maxNumAppelli];
57    Prenotazione prenotazione;
58  } RichiestaSegreteria;
59
60  typedef struct RispostaServer_str{
61    bool errore;
62    int numAppelli;
63  }RispostaServer;
64  #endif
```

Alcune strutture dati sono allocate in maniera dinamica, specialmente sull'applicativo segreteria, in modo da non riempire la memoria nel caso di lungo utilizzo. Un esempio può essere la creazione di un messaggio di richiesta tra la segreteria e il server universitario, la cui creazione e distruzione è mostrato nel listing 3.

Listing 3: Invia esami al server

```
1  printf("Invio gli esami al server\n");
```

```
2  RichiestaSegreteria * richiesta = (RichiestaSegreteria *)malloc(sizeof(↩
       RichiestaSegreteria));
3  RispostaServer * risposta = (RispostaServer *)malloc(sizeof(RispostaServer↩
       ));
4
5  richiesta->numAppelli = numAppelli;
6  richiesta->type = 1;
7  memcpy(richiesta->appelli, appelli, sizeof(Appello)*maxNumAppelli);
8  FullWrite(socket, (void*)&(*richiesta), sizeof(RichiestaSegreteria));
9  FullRead(socket, &(*risposta), sizeof(RispostaServer));
10 if(risposta->errore == 0){
11     printf("Caricati %d esami\n", risposta->numAppelli);
12 }else{
13     printf("Errore caricamento esami\n");
14 }
15 free(richiesta);
16 free(risposta);
```

Utilizzo

Per eseguire correttamente il codice è necessario compilare prima tutti e tre gli eseguibili, con i comandi del listing 4.

Listing 4: Compilazione Eseguibili.

```
1 gcc server.c FullWrite.c FullRead.c -o server
2 gcc segreteria.c FullWrite.c FullRead.c -o segreteria
3 gcc studente.c FullWrite.c FullRead.c -o studente
```

Una volta compilato, è necessario avviare gli eseguibili **nel seguente ordine**: 1) server, 2) segreteria, 3) studente in tre terminali diversi. Il processo è mostrato nel listing 5.

Listing 5: Compilazione Eseguibili.

```
1 ./server
2 ./segreteria
3 ./studente
```