Reti di Calcolatori – 9 luglio 2019

Si progetti un'applicazione Client/Server che, utilizzando le socket, permetta a un appassionato bevitore di caffé di consultare le informazioni sulle macchine da caffé presenti sul mercato. L'applicazione deve presentare la seguente interfaccia:

coffee_machines server porta

dove ***server*** rappresenta il nome logico del Server e ***porta*** rappresenta il numero di porta del servizio. Per prima cosa, il Client si deve interfacciare con l'utente, da cui riceve (via terminale) *username*, *password*, e la *categoria di macchine* di interesse (es., "superautomatiche", "capsule", "cialde", "pour over", "cold brew", ecc.). Il Client deve quindi trasmettere le informazioni al Server, che a sua volta dovrà verificare l'autorizzazione dell'utente invocando un'apposita funzione, che si suppone essere già implementata con il seguente prototipo:

```
int autorizza(const char *username, const char *password);
```

Se la funzione `autorizza` restituisce il valore 1, l'utente è autorizzato ad accedere al servizio e il Server dovrà quindi: analizzare tutte le informazioni sulle macchine da caffé di interesse all'interno del proprio database; selezionare tra queste solo le informazioni relative alla media di recensione, al nome del modello e al prezzo; elencare le informazioni in ordine di media di recensione decrescente (ovverosia dalla media più elevata a quella più bassa); considerare solo le prime 10 (diconsi dieci) macchine in elenco; e infine restituire il risultato al Client. Nel caso l'utente non fosse autorizzato, il Server dovrà rifiutarsi di eseguire il servizio e inviare un messaggio di errore al Client.

A questo proposito, si supponga che le informazioni sulle macchine per caffé disponibili in commercio siano salvate sul Server in una serie di file di testo all'interno del percorso */var/local/macchine_caffé/,* ciascuno dei quali conterrà le informazioni per una specifica tipologia di macchine. (Quindi, per esempio, le informazioni sulle macchine superautomatiche saranno salvate nel file "*/var/local/macchine_caffé/superautomatiche.txt*", ecc.) Ciascuna riga di tale file conterrà tutte le informazioni relative a una singola macchina, con (in quest'ordine) media di recensione (che si suppone essere un numero intero da 0 a 100), nome del produttore, nome del modello, prezzo, ecc.

Una volta ricevute le informazioni dal Server, il Client le stampa a video e si mette in attesa della richiesta successiva. Il Client deve terminare quando l'utente digita "fine".

ATTENZIONE: Si realizzino il Client e il Server in C, ma il Client deve essere realizzato anche in Java.

```c
// Server esame 2019-07-09

#include <sys/socket.h>
#include <sys/wait.h>
#define _POSIX_C_SOURCE 200809L
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netdb.h>
#include <unistd.h>
#include <signal.h>
#include <errno.h>
#include "rxb.h"
#include "utils.h"

/* Massima lunghezza stringhe: 64 KiB */
#define MAX_REQUEST_SIZE 4096

/* Implementazione funzione autorizza */
int autorizza(const char *username, const char *password)
{
    return 1;
}

/* Gestore del segnale SIGCHLD */
void handler(int signo)
{
        int status;

        (void)signo; /* per evitare warning */

        /* eseguo wait non bloccanti finché ho dei figli terminati */
        while (waitpid(-1, &status, WNOHANG) > 0)
                continue;
}

/* argv[0] argv[1] */
/* server  porta */
int main (int argc, char *argv[])
{
    int err, sd, on;
    struct addrinfo hints, *res;
        struct sigaction sa = { 0 };

    /* Controllo argomenti */
    if (argc != 2) {
        exit(EXIT_FAILURE);
    }

        sigemptyset(&sa.sa_mask);
        /* uso SA_RESTART per evitare di dover controllare esplicitamente se
         * accept è stata interrotta da un segnale e in tal caso rilanciarla
         * (si veda il paragrafo 21.5 del testo M. Kerrisk, "The Linux
         * Programming Interface") */
        sa.sa_flags   = SA_RESTART;
        sa.sa_handler = handler;

        if (sigaction(SIGCHLD, &sa, NULL) == -1) {
                perror("sigaction");
                exit(EXIT_FAILURE);
        }

    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;

    err = getaddrinfo(NULL, argv[1], &hints, &res);
    if (err != 0) {
        exit(EXIT_FAILURE);
    }
```

```
72
73      sd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
74      if (sd < 0) {
75          exit(EXIT_FAILURE);
76      }
77
78          on = 1;
79          if (setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on)) < 0){
80                  perror("setsockopt");
81                  exit(EXIT_FAILURE);
82          }
83
84      err = bind(sd, res->ai_addr, res->ai_addrlen);
85      if (err < 0) {
86          exit(EXIT_FAILURE);
87      }
88
89      err = listen(sd, SOMAXCONN);
90      if (err < 0) {
91          exit(EXIT_FAILURE);
92      }
93
94      while (1) {
95          int ns, pid_f;
96
97          ns = accept(sd, NULL, NULL);
98          if (ns < 0) {
99              if (errno == EINTR) {
100                 continue;
101             } else {
102                 exit(EXIT_FAILURE);
103             }
104         }
105
106         pid_f = fork();
107         if (pid_f < 0) {
108             exit(EXIT_FAILURE);
109         } else if (pid_f == 0) {
110             /* FIGLIO */
111             rxb_t rxb;
112                         char username[MAX_REQUEST_SIZE];
113                         char password[MAX_REQUEST_SIZE];
114                         char categoria[MAX_REQUEST_SIZE];
115             size_t username_len, password_len, categoria_len;
116
117             close(sd);
118
119             /* creo buffer rxb */
120             rxb_init(&rxb, MAX_REQUEST_SIZE * 2);
121
122             /* disinstallo gestore segnali */
123             signal(SIGCHLD, SIG_DFL);
124
125             /* ciclo di richieste */
126             while (1) {
127                 char *end_request = "--- END REQUEST ---\n";
128                 int pid_cut, pid_sort, pid_head, status;
129                 int pipe_cs[2], pipe_sh[2];
130
131                 /* leggere 3 informazioni */
132                 memset(username, 0, sizeof(username));
133                 username_len = sizeof(username) - 1;
134                 err = rxb_readline(&rxb, ns, username, &username_len);
135                 if (err < 0) {
136                     exit(EXIT_FAILURE);
137                 }
138
139                 memset(password, 0, sizeof(password));
140                 password_len = sizeof(password) - 1;
141                 err = rxb_readline(&rxb, ns, password, &password_len);
142                 if (err < 0) {
```

```
143                     exit(EXIT_FAILURE);
144                 }
145
146                 memset(categoria, 0, sizeof(categoria));
147                 categoria_len = sizeof(categoria) - 1;
148                 err = rxb_readline(&rxb, ns, categoria, &categoria_len);
149                 if (err < 0) {
150                     exit(EXIT_FAILURE);
151                 }
152
153                 /* chiamare autorizza */
154                 if (autorizza(username, password) != 1) {
155                     char *unauthorized = "Non autorizzato!\n";
156                     write_all(ns, unauthorized, strlen(unauthorized));
157                     write_all(ns, end_request, strlen(end_request));
158                     continue;
159                 }
160
161                 if (pipe(pipe_cs) < 0) {
162                     exit(EXIT_FAILURE);
163                 }
164
165                 /* creo nipoti */
166                 pid_cut = fork();
167                 if (pid_cut < 0) {
168                     exit(EXIT_FAILURE);
169                 } else if (pid_cut == 0) {
170                     char nomefile[MAX_REQUEST_SIZE + 256];
171
172                     /* NIPOTE CUT */
173                     close(ns);
174                     close(1);
175                     dup(pipe_cs[1]);
176                     close(pipe_cs[1]);
177                     close(pipe_cs[0]);
178
179                     /*
180                     snprintf(nomefile, sizeof(nomefile),
181                         "/var/local/macchine_caffé/%s.txt", categoria);
182                     */
183                     snprintf(nomefile, sizeof(nomefile),
184                         "%s.txt", categoria);
185                     /* strcpy(nomefile, categoria); strcat(nomefile, ".txt"); */
186
187                     execlp("cut", "cut", "-d", ",", "-f", "1,3,4", nomefile, NULL);
188                     perror("exec");
189                     exit(EXIT_FAILURE);
190                 }
191
192                 if (pipe(pipe_sh) < 0) {
193                     exit(EXIT_FAILURE);
194                 }
195
196                 pid_sort = fork();
197                 if (pid_sort < 0) {
198                     exit(EXIT_FAILURE);
199                 } else if (pid_sort == 0) {
200                     /* NIPOTE SORT */
201                     close(ns);
202
203                     close(0);
204                     dup(pipe_cs[0]);
205                     close(pipe_cs[1]);
206                     close(pipe_cs[0]);
207
208                     close(1);
209                     dup(pipe_sh[1]);
210                     close(pipe_sh[1]);
211                     close(pipe_sh[0]);
212
213                     execlp("sort", "sort", "-rn", NULL);
```

```
214                     perror("exec");
215                     exit(EXIT_FAILURE);
216                 }
217 
218                 pid_head = fork();
219                 if (pid_head < 0) {
220                     exit(EXIT_FAILURE);
221                 } else if (pid_head == 0) {
222                     /* NIPOTE HEAD */
223                     close(pipe_cs[1]);
224                     close(pipe_cs[0]);
225 
226                     close(0);
227                     dup(pipe_sh[0]);
228                     close(pipe_sh[1]);
229                     close(pipe_sh[0]);
230 
231                     close(1);
232                     dup(ns);
233                     close(ns);
234 
235                     execlp("head", "head", "-n", "10", NULL);
236                     perror("exec");
237                     exit(EXIT_FAILURE);
238                 }
239 
240                 /* chiudo pipe che non use il figlio */
241                 close(pipe_cs[1]);
242                 close(pipe_cs[0]);
243                 close(pipe_sh[1]);
244                 close(pipe_sh[0]);
245 
246                 /* aspetto terminazione nipoti */
247                 waitpid(pid_cut, &status, 0);
248                 waitpid(pid_sort, &status, 0);
249                 waitpid(pid_head, &status, 0);
250 
251                 /* mando stringa fine richiesta */
252                 write_all(ns, end_request, strlen(end_request));
253             }
254 
255             /* cleanup */
256             close(ns);
257             rxb_destroy(&rxb);
258             exit(EXIT_SUCCESS);
259         }
260 
261         /* PADRE */
262         close(ns);
263     }
264 
265     close(sd);
266 
267     return 0;
268 }
269 
```

```c
// Client C esame 2019-07-09

#include <sys/socket.h>
#define _POSIX_C_SOURCE 200809L
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "utils.h"
#include "rxb.h"

#define DIM 80
#define MAX_REQUEST_SIZE 4096

/*
 *         argv[1] argv[2]
 * coffee_machines server  porta
 */
int main (int argc, char *argv[])
{
    int err, sd;
    struct addrinfo hints, *res, *ptr;
    rxb_t rxb;

    if (argc != 3) {
        fprintf(stderr, "Usage: coffee_machines server porta\n");
        exit(EXIT_FAILURE);
    }

    /* chiamata DNS */
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;

    err = getaddrinfo(argv[1], argv[2], &hints, &res);
    if (err != 0) {
        exit(EXIT_FAILURE);
    }

    /* procedura di connessione */
    for (ptr = res; ptr != NULL; ptr = ptr->ai_next) {
        sd = socket(ptr->ai_family, ptr->ai_socktype, ptr->ai_protocol);
        if (sd < 0)
            continue;

        err = connect(sd, ptr->ai_addr, ptr->ai_addrlen);
        if (err == 0)
            break;

        close(sd);
    }

    if (ptr == NULL) {
        exit(EXIT_FAILURE);
    }

    freeaddrinfo(res);

    rxb_init(&rxb, MAX_REQUEST_SIZE * 2);

    /* ciclo di servizio */
    while (1) {
        char username[DIM];
        char password[DIM];
        char categoria_macchine[DIM];
        char line[4096];

        puts("Inserisci username:");
        if (fgets(username, sizeof(username), stdin) == NULL) {
            exit(EXIT_FAILURE);
```

```
72          }
73
74          puts("Inserisci password:");
75          if (fgets(password, sizeof(password), stdin) == NULL) {
76              exit(EXIT_FAILURE);
77          }
78
79          puts("Inserisci categoria macchine:");
80          if (fgets(categoria_macchine, sizeof(categoria_macchine), stdin) == NULL) {
81              exit(EXIT_FAILURE);
82          }
83
84          if (write_all(sd, username, strlen(username)) < 0) {
85              exit(EXIT_FAILURE);
86          }
87
88          if (write_all(sd, password, strlen(password)) < 0) {
89              exit(EXIT_FAILURE);
90          }
91
92          if (write_all(sd, categoria_macchine, strlen(categoria_macchine)) < 0) {
93              exit(EXIT_FAILURE);
94          }
95
96          while (1) {
97              size_t buflen = sizeof(line)-1;
98
99              memset(line, 0, sizeof(line));
100
101             err = rxb_readline(&rxb, sd, line, &buflen);
102             if (err < 0) {
103                 exit(EXIT_FAILURE);
104             }
105
106             if (strcmp(line, "--- END REQUEST ---") == 0)
107                 break;
108
109             puts(line);
110             /* printf("%s\n", line); */
111             /* write_all(STDOUT_FILENO, line, buflen); write(STDOUT_FILENO, "\n",
                1); */
112         }
113
114     }
115
116     rxb_destroy(&rxb);
117
118     close(sd);
119
120     return 0;
121 }
122
```

```java
// Client Java esame 2019-07-09

import java.io.*;
import java.net.*;

public class ClientTDConnreuse
{
    public static void main(String[] args) {
        if (args.length != 2) {
            System.err.println("Usage: java ClientTDConnreuse server porta");
            System.exit(1);
        }

        try {
            Socket s = new Socket(args[0], Integer.parseInt(args[1]));
            BufferedReader netIn  = new BufferedReader(new
            InputStreamReader(s.getInputStream(), "UTF-8"));
            BufferedWriter netOut = new BufferedWriter(new
            OutputStreamWriter(s.getOutputStream(), "UTF-8"));
            BufferedReader keyboard = new BufferedReader(new
            InputStreamReader(System.in));

            while (true) {
                System.out.println("Inserisci username:");
                String username = keyboard.readLine();

                System.out.println("Inserisci password:");
                String password = keyboard.readLine();

                System.out.println("Inserisci categoria macchine:");
                String categoria = keyboard.readLine();

                netOut.write(username); netOut.newLine();
                netOut.write(password); netOut.newLine();
                netOut.write(categoria); netOut.newLine();
                netOut.flush();

                String line;
                while(true) {
                    line = netIn.readLine();
                    if (line == null) {
                        s.close();
                        System.exit(0);
                    }
                    if (line.equals("--- END REQUEST ---")) {
                        break;
                    }
                    System.out.println(line);
                }
            }

            // s.close();
        }
        catch (Exception e) {
            System.err.println(e.getMessage());
            e.printStackTrace();
            System.exit(1);
        }
    }
}

```