Architetture di Reti / Reti di Calcolatori – 14 gennaio 2021

Si progetti un'applicazione Client/Server che, utilizzando le socket, permetta al titolare di un'azienda di consultare le vendite effettuate a clienti provenienti da una nazione di interesse in un mese (e anno) di interesse. L'applicazione deve presentare la seguente interfaccia:

browse_sales 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) il *mese di interesse* (es., "gennaio", ecc.), l'*anno di interesse* (es., "2019", "2020", ecc.), la *nazione di provenienza del cliente* di interesse (es., "Italia", "Germania", "Regno Unito", ecc.). Il Client deve quindi trasmettere le informazioni al Server, che a sua volta dovrà reperire le informazioni sulle vendite effettuate ai clienti della nazione di provenienza di interesse nel periodo di interesse, elencarle in ordine di numero di unità di prodotto decrescente (ovverosia dalla vendita con il numero di unità di prodotto maggiore a quella con il numero di unità di prodotto minore), e restituirle al Client. Opzionalmente, il Server dovrà anche calcolare i ricavi [1] ottenuti da ciascuna delle vendite di interesse e inviare al Client (solo) il ricavo maggiore tra quelli esaminati.

A questo proposito, si supponga che le informazioni sulle spese sostenute dall'azienda siano salvate sul Server in una serie di file di testo all'interno del percorso */var/local/sales*, organizzati per mese e anno. (Quindi, per esempio, le informazioni sulle vendite effettuate nel dicembre 2020 saranno salvate nel file */var/local/sales/2020/dicembre.txt*.) Ciascuna riga di tali file conterrà tutte le informazioni relative a un singolo prodotto, con (in quest'ordine) il numero di unità di prodotto vendute, il costo unitario del prodotto, il codice del prodotto, il codice cliente, la nazione di provenienza del cliente, 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 Server in C/Unix e il Client sia in C/Unix che in Java.

[1] Si noti che una vendita riguarda un singolo prodotto ma può coinvolgere più unità di quel prodotto. Quindi il ricavo della vendita è rappresentato dal numero di unità vendute per il costo unitario del prodotto.

```c
//Server esame 2021-01-14

#define _POSIX_C_SOURCE 200809L
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <string.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <unistd.h>
#include <limits.h>
#ifdef USE_LIBUNISTRING
#include <unistr.h>
#endif
#include "rxb.h"
#include "utils.h"

#define MAX_LINE 256
#define MAX_REQUEST_SIZE (64 * 1024)

/* Funzione che usa strtol per leggere il numero naturale (ovverosia intero non
 * negativo) all'inizio di una stringa. */
int get_natural(const char *str)
{
        long ret;
        char *endptr;

        ret = strtol(str, &endptr, 10);

        if (ret == 0 && errno == EINVAL) {
                // nessuna conversione effettuata
                return -1;
        }

        if (errno == ERANGE) {
                if (ret == LONG_MIN) {
                        // underflow
                        return -2;
                } else { // ret == LONG_MAX
                        // overflow
                        return -3;
                }
        }

        if (ret < 0 || ret > INT_MAX) {
                return -4;
        }

        return (int)ret;
}

/* Funzione che usa strtof per leggere il numero reale positivo all'inizio di
 * una stringa. */
float get_positive_real(const char *str)
{
        float ret;
        char *endptr;

        ret = strtof(str, &endptr);

    /* Nessuna conversione effettuata */
        if (endptr == str) {
                return -1;
        }

    /* Fuori range */
```

```
72          if (errno == ERANGE) {
73          return -2;
74          }
75
76      /* Scarto valori negativi */
77          if (ret < 0.0) {
78                  return -3;
79          }
80
81          return ret;
82  }
83
84  void max_ricavo(int fd_in, int fd_out )
85  {
86      char *col1, *col2;
87      char result[MAX_REQUEST_SIZE];
88      char line[MAX_LINE];
89      float ricavo, ricavo_max = 0.0;
90      float prezzo_unitario;
91      int pezzi_venduti;
92      FILE *fp;
93
94      /* Apro un FILE per leggere l'input in modo bufferizzato. Nonostante
95       * questa tecnica non sia adatta per leggere l'input da una socket, nel
96       * nostro caso va benissimo per processare l'input proveniente da una
97       * pipe. */
98      fp = fdopen(fd_in, "r");
99      if (fp == NULL) {
100         fprintf(stderr, "Errore apertura fp");
101         exit(EXIT_FAILURE);
102     }
103
104     while (fgets(line, sizeof(line), fp) != NULL) {
105         /* Scrivo la riga letta sull'output */
106         if (write_all(fd_out, line, strlen(line)) < 0) {
107             perror("write_all");
108             exit(EXIT_FAILURE);
109         }
110
111         /* Assumo un separatore ; tra colonne. Uso strtok(line, ";") per
112          * ottenere il testo della prima colonna e strtok(NULL, ";") per
113          * ottenere quello della colonna successiva.
114          * NB: strtok è una funzione non rientrante ma il suo uso in un
115          * contesto come questo è più che accettabile. */
116         col1 = strtok(line, ";");
117         pezzi_venduti = get_natural(col1);
118
119         col2 = strtok(NULL, ";");
120         prezzo_unitario = get_positive_real(col2);
121
122         ricavo = (float)pezzi_venduti * prezzo_unitario;
123
124         if (ricavo > ricavo_max) {
125             ricavo_max = ricavo;
126         }
127     }
128
129     /* Chiudo il FILE fp (e il corrispondente file descriptor fd_in) */
130     fclose(fp);
131
132     /* Scrivo ricavo massimo */
133     snprintf(result, sizeof(result), "Ricavo massimo: %f\n", ricavo_max);
134     if (write_all(fd_out, result, strlen(result)) < 0) {
135         perror("write");
136         exit(EXIT_FAILURE);
137     }
138 }
139
140 void handler(int signo)
141 {
142     int status;
```

```c

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

    while (waitpid(-1, &status, WNOHANG) > 0)
        continue;
}

int main(int argc, char **argv)
{
    struct addrinfo hints, *res;
    int sd, ns, err, pid;
    struct sigaction sa;
    int optval = 1;

    if (argc != 2) {
        fprintf(stderr, "Usage: %s port\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    memset(&sa, 0, sizeof(sa));
    sigemptyset(&sa.sa_mask);
    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;

    if ((err = getaddrinfo(NULL, argv[1], &hints, &res)) != 0) {
        fprintf(stderr, "Errore setup indirizzo bind: %s\n", gai_strerror(err));
        exit(EXIT_FAILURE);
    }

    if ((sd = socket(res->ai_family, res->ai_socktype, res->ai_protocol)) < 0) {
        perror("Errore in socket");
        exit(EXIT_FAILURE);
    }

    if (setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval)) < 0) {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }

    if (bind(sd, res->ai_addr, res->ai_addrlen) < 0) {
        perror("Errore in bind");
        exit(EXIT_FAILURE);
    }

    freeaddrinfo(res);

    if (listen(sd, SOMAXCONN) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }

    for (;;) {
        if ((ns = accept(sd, NULL, NULL)) < 0) {
            perror("Errore in accept");
            exit(EXIT_FAILURE);
        }

        if ((pid = fork()) < 0) {
            perror("fork");
            exit(EXIT_FAILURE);
        } else if (pid == 0) { // FIGLIO
```

```
214             const char *end_request = "\n---END REQUEST---\n";
215             rxb_t rxb;
216
217             close(sd);
218
219             rxb_init(&rxb, MAX_REQUEST_SIZE);
220
221             /* Ciclo di gestione delle richieste */
222             for (;;)
223             {
224                 int pid2, pid3, pipe_nf[2], pipe_pnn[2];
225                 char mese[MAX_REQUEST_SIZE];
226                 char anno[MAX_REQUEST_SIZE];
227                 char nazione[MAX_REQUEST_SIZE];
228                 size_t mese_len, anno_len, nazione_len;
229
230                 memset(mese, 0, sizeof(mese));
231                 mese_len = sizeof(mese) - 1;
232
233                 if (rxb_readline(&rxb, ns, mese, &mese_len) < 0) {
234                     rxb_destroy(&rxb);
235                     break;
236                 }
237
238 #ifdef USE_LIBUNISTRING
239                 if (u8_check((uint8_t *)mese, mese_len) != NULL) {
240                     fprintf(stderr, "Request is not valid UTF-8");
241                     close(ns);
242                     exit(EXIT_SUCCESS);
243                 }
244 #endif
245
246                 memset(anno, 0, sizeof(anno));
247                 anno_len = sizeof(anno) - 1;
248
249                 if (rxb_readline(&rxb, ns, anno, &anno_len) < 0) {
250                     rxb_destroy(&rxb);
251                     break;
252                 }
253
254 #ifdef USE_LIBUNISTRING
255                 if (u8_check((uint8_t *)anno, anno_len) != NULL) {
256                     fprintf(stderr, "Request is not valid UTF-8");
257                     close(ns);
258                     exit(EXIT_SUCCESS);
259                 }
260 #endif
261
262                 memset(nazione, 0, sizeof(nazione));
263                 nazione_len = sizeof(nazione) - 1;
264
265                 if (rxb_readline(&rxb, ns, nazione, &nazione_len) < 0) {
266                     rxb_destroy(&rxb);
267                     break;
268                 }
269
270 #ifdef USE_LIBUNISTRING
271                 if (u8_check((uint8_t *)nazione, nazione_len) != NULL) {
272                     fprintf(stderr, "Request is not valid UTF-8");
273                     close(ns);
274                     exit(EXIT_SUCCESS);
275                 }
276 #endif
277                 /* Creo la pipe tra filgio pid e nipote pid1 per fare la redirezione
278                    dell'output della sort verso la funzione per calcolare
279                    il ricavo massimo (devo passare un descrittore) */
280                 if (pipe(pipe_nf) < 0) {
281                     perror("pipe_nf");
282                     exit(EXIT_FAILURE);
283                 }
284
```

```
285                 if ((pid2 = fork()) < 0) {
286                     perror("fork");
287                     exit(EXIT_FAILURE);
288                 } else if (pid2 == 0) { // NIPOTE che si occuperà dell'esecuzione
                    della sort
289                     close(ns);
290                     /* Credo la pipe tra nipote pid2 e pronipote pid3 per passare
                        l'output
291                     della grep come input della sort */
292                     if (pipe(pipe_pnn)) {
293                         perror("pipe");
294                         exit(EXIT_FAILURE);
295                     }
296
297                     if ((pid3 = fork()) < 0) {
298                         perror("terza fork");
299                         exit(EXIT_FAILURE);
300                     } else if (pid3 == 0) { /* PRONIPOTE */
301                         char filename[2 * MAX_REQUEST_SIZE + 256];
302
303                         //chiudo le pipe che non mi servono
304                         close(pipe_nf[0]); //descrittore uscita pipe_nf
305                         close(pipe_nf[1]); //descrittore ingresso pipe_nf
306                         close(pipe_pnn[0]); //descrittore uscita pipe_pnn
307
308                         close(1); //chiudo stdout
309                         if (dup(pipe_pnn[1]) < 0) { //con dup reindirizzo stdout
                            come ingresso di pipe_pnn[1]
310                             perror("dup");
311                             exit(EXIT_FAILURE);
312                         }
313                         close(pipe_pnn[1]);
314
315                         //creo la file path completa da passare poi come parametro
                            alla grep
316                         snprintf(filename, sizeof(filename), "%s/%s.txt", anno, mese);
317                         // snprintf(filename, sizeof(filename),
                            "/var/local/sales/%s/%s.txt",anno,mese);
318
319                         //filtro i dati in base alla nazione
320                         execlp("grep", "grep", nazione, filename, (char *)NULL);
321                         perror("execlp grep");
322                         exit(EXIT_FAILURE);
323                     }
324
325                     // chiudo le pipe che non mi servono
326                     close(pipe_pnn[1]); //descrittore ingresso pipe_pnn[1]
327                     close(pipe_nf[0]); //&descrittore uscita pipe_nf[0]
328
329                     close(0); //chiudo stdin
330                     if (dup(pipe_pnn[0]) < 0) { //con dup reindirizzo su stdin
                        l'estremo d'uscita della pipe_pnn[0]
331                         perror("dup");
332                         exit(EXIT_FAILURE);
333                     }
334                     close(pipe_pnn[0]);
335
336                     close(1); //chiudo stdout
337                     if (dup(pipe_nf[1]) < 0) { //con dup reindirizzo stdout come
                        ingresso di pipe_nf[1]
338                         perror("dup");
339                         exit(EXIT_FAILURE);
340                     }
341                     close(pipe_nf[1]);
342
343                     //ordino i dati ricevuti dalla grep in ordine decrescente e
                        numerico
344                     execlp("sort", "sort", "-rn", (char *)NULL);
345                     perror("execlp sort");
346                     exit(EXIT_FAILURE);
347                 }
```

```
348
349                 // FIGLIO
350                 close(pipe_nf[1]); //descrittore ingresso pipe_nf
351
352                 /* Trovo ricavo massimo
353                    Passo alla funzione l'estremo d'uscita della pipe (da dove
                       leggere l'output della sort)
354                    e il socket descriptor da cui stampare a video riga per riga e il
                       ricavo massimo*/
355                 max_ricavo(pipe_nf[0], ns);
356
357                 /* Trasmetto messaggio fine richiesta */
358                 if (write_all(ns, end_request, strlen(end_request)) < 0) {
359                     perror("write");
360                     exit(EXIT_FAILURE);
361                 }
362             }
363
364             close(ns); //chiudo la socket attiva nel processo figlio
365
366             exit(EXIT_SUCCESS);
367         }
368
369         close(ns); //chiudo la socket attiva nel loop e passo ad un'altra richiesta
370     }
371
372     close(sd); //chiudo la socket passiva e termino il programma
373
374     return 0;
375 }
376
377
```

```c
//Client c esame 2021-01-14

#define _POSIX_C_SOURCE 200809L

#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <string.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include "rxb.h"
#include "utils.h"
#include <unistd.h>

#ifdef USE_LIBUNISTRING
#include <unistr.h>
#endif

#define MAX_REQUEST_SIZE (64 * 1024)

int main(int argc, char const *argv[])
{
    int sd, err;
    struct addrinfo hints, *ptr, *res;
    rxb_t rxb;

    char mese[MAX_REQUEST_SIZE];
    char anno[MAX_REQUEST_SIZE];
    char nazione[MAX_REQUEST_SIZE];
    char response[MAX_REQUEST_SIZE];
    size_t response_len;

    if (argc != 3) {
        fprintf(stderr, "Sintassi: %s hostname port", argv[0]);
        exit(EXIT_FAILURE);
    }

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

    if ((err = getaddrinfo(argv[1], argv[2], &hints, &res)) != 0) {
        fprintf(stderr, "Errore di risoluzione nome: %s \n", gai_strerror(err));
        exit(EXIT_FAILURE);
    }

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

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

        close(sd);
    }

    if (ptr == NULL) {
        fprintf(stderr, "Errore di connesione! \n");
        exit(EXIT_FAILURE);
    }

    freeaddrinfo(res);

    rxb_init(&rxb, MAX_REQUEST_SIZE);
```

```
72      for (;;) {
73          puts("Inserire mese (fine per terminare): ");
74          if (fgets(mese, sizeof(mese), stdin) == NULL) {
75              perror("fgets");
76              exit(EXIT_FAILURE);
77          }
78
79          if (strcmp(mese, "fine\n") == 0) {
80              break;
81          }
82
83          puts("Inserire anno: ");
84          if (fgets(anno, sizeof(anno), stdin) == NULL) {
85              perror("fgets");
86              exit(EXIT_FAILURE);
87          }
88
89          puts("Inserire nazione: ");
90          if (fgets(nazione, sizeof(nazione), stdin) == NULL) {
91              perror("fgets");
92              exit(EXIT_FAILURE);
93          }
94
95          if (write_all(sd, mese, strlen(mese)) < 0) {
96              perror("write");
97              exit(EXIT_FAILURE);
98          }
99
100         if (write_all(sd, anno, strlen(anno)) < 0) {
101             perror("write");
102             exit(EXIT_FAILURE);
103         }
104
105         if (write_all(sd, nazione, strlen(nazione)) < 0) {
106             perror("write");
107             exit(EXIT_FAILURE);
108         }
109
110         for (;;) {
111             memset(response, 0, sizeof(response));
112             response_len = sizeof(response) - 1;
113
114             if (rxb_readline(&rxb, sd, response, &response_len) < 0) {
115                 rxb_destroy(&rxb);
116                 fprintf(stderr, "Connessione chiusa dal server! \n");
117                 exit(EXIT_FAILURE);
118             }
119
120             puts(response);
121
122             if (strcmp(response, "---END REQUEST---") == 0) {
123                 break;
124             }
125         }
126     }
127
128     close(sd);
129
130     return 0;
131 }
132
```

```java
// Client Java esame 2021-01-14

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

class ClientTDConnreuse {

    public static void main(String[] args) {

        try {
            if (args.length != 2) {
                System.err.println("Usage: java ClientTDConnreuse hostname port");
                System.exit(1);
            }

            Socket s = new Socket(args[0], Integer.parseInt(args[1]));
            BufferedReader userIn = new BufferedReader(new
            InputStreamReader(System.in));
            BufferedReader networkIn = new BufferedReader(new
            InputStreamReader(s.getInputStream(), "UTF-8"));
            BufferedWriter networkOut = new BufferedWriter(new
            OutputStreamWriter(s.getOutputStream(), "UTF-8"));

            for (;;) {

                System.out.print("\nInserire mese (fine per terminare):");
                String mese = userIn.readLine();

                if (mese.equals("fine")) {
                    break;
                }

                System.out.print("\nInserire anno:");
                String anno = userIn.readLine();

                System.out.print("\nInserire nazione:");
                String nazione = userIn.readLine();

                networkOut.write(mese);
                networkOut.newLine();
                networkOut.flush();

                networkOut.write(anno);
                networkOut.newLine();
                networkOut.flush();

                networkOut.write(nazione);
                networkOut.newLine();
                networkOut.flush();

                String input;

                for (;;) {
                    if ((input = networkIn.readLine()) == null) {

                        System.err.println("Errore! Il Server ha chiuso la
                        connessione!");
                        System.exit(2);
                    }

                    System.out.println(input);

                    if (input.equals("---END REQUEST---"))
                        break;
                }
            }

            s.close();

        } catch (Exception e) {
            System.err.println(e.getMessage());
```

```
68                  e.printStackTrace();
69                  System.exit(3);
70              }
71          }
72      }
73
```