Reti di Calcolatori – 20 dicembre 2021

Si progetti un'applicazione Client/Server che, utilizzando le socket, permetta a un utente in ritardo con l'acquisto dei regali di cercare articoli disponibili per l'acquisto "last minute" tra quelli appartenenti a una tipologia di interesse. L'applicazione deve presentare la seguente interfaccia:

regali_last_minute 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) la *categoria di regali* (es., "giocattoli", "elettronica", ecc.), il *nome del produttore* (es., "LEGO", "Apple", ecc.), e la *modalità di ordinamento* relativamente al prezzo dell'articolo (che potrà quindi essere "crescente" o "decrescente") di interesse. Il Client deve quindi trasmettere le informazioni al Server, che a sua volta dovrà reperire le informazioni sugli articoli di interesse e restituirle al Client nell'ordine desiderato.

A questo proposito, si supponga che le informazioni sugli articoli disponibili per l'acquisto last minute siano salvate sul Server in una serie di file di testo all'interno del percorso */var/local/last_minute_gifts*[1], organizzati per categoria di regalo. (Quindi, per esempio, le informazioni sui giocattoli saranno salvate nel file */var/local/last_minute_gifts/giocattoli.txt*.) Ciascuna riga di tali file conterrà tutte le informazioni relative a un singolo articolo disponibile per l'acquisto last minute, con (in quest'ordine) il prezzo, il nome del produttore, il nome dell'articolo, il codice dell'articolo, 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 linguaggio C e il Client sia in linguaggio C che in linguaggio Java.

---

[1] Ovviamente nei PC del laboratorio non avrete permessi di accesso al percorso /var. Ai fini dell'esame, potete utilizzare un percorso all'interno della vostra home e lasciare un opportuno commento nella soluzione dell'esercizio (es. "uso il percorso ./last_minute_gifts al posto di /var/local/last_minute_gifts").

```c
//Server esame 2021-12-20

#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 <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include "utils.h"
#include "rxb.h"

#define MAX_REQUEST_SIZE 4096

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

    (void)signo;

    /*gestisco tutti i figlio terminati*/
    while (waitpid(-1, &status, WNOHANG) > 0)
    {
        continue;
    }
}

int main(int argc, char **argv){
    struct addrinfo hints, *res;
    int err, sd, ns, pid, status;
    struct sigaction sa;
    char *porta = argv[1]; //argv[1] è l'argomento passato da terminale dove è memorizzato il numero della porta
    int optval = 1;

    if (argc != 2)
    {
        perror("Argomenti non validi! Ugtilizzo: ./server <porta>");
        exit(EXIT_FAILURE);
    }

    memset(&sa, 0, sizeof(sa));
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = SA_RESTART;
    sa.sa_handler = handler;

    //Installo il controllo del segnale SIGCHLD con la funzione handler
    if (sigaction(SIGCHLD, &sa, NULL) == -1)
    {
        perror("sigaction");
        exit(EXIT_FAILURE);
    }

    //Inizializzo la struttura hints per la getaddrinfo()
    memset(&hints, 0, sizeof(hints));

    //Imposto le proprietà della socket
    hints.ai_family = AF_UNSPEC;     //Socket funzionante sia per IPv4 che per IPv6
    hints.ai_socktype = SOCK_STREAM; //Uso una socket con connessione e affidabile --> TCP
    hints.ai_flags = AI_PASSIVE;     //Socket passiva (caso server);

    //Utilizzo getaddrinfo() per preparare la socket
    if ((err = getaddrinfo(NULL, porta, &hints, &res)) != 0){
```

```
70          /* la funzione gai_strerror(err) restituisce una stringa che spiega il
71          codice errore della getaddrinfo() */
72          fprintf(stderr, "Errore setup indirizzo bind: %s\n", gai_strerror(err));
73          exit(EXIT_FAILURE);
74      }
75
76      //Creo la socket passiva
77      //Uso la prima struttura contenuta nella lista puntata da res
78      if ((sd = socket(res->ai_family, res->ai_socktype, res->ai_protocol)) < 0){
79          perror("socket");
80          exit(EXIT_FAILURE);
81      }
82
83      if (setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval)) < 0) {
84          perror("setsockopt");
85          exit(EXIT_FAILURE);
86      }
87
88      //Attraverso la bind associo la socket all'indirizzo locale
89      //Passo come argomenti la struttura contenuta nella lista di struttura addrinfo
        puntate da res
90      if (bind(sd, res->ai_addr, res->ai_addrlen) < 0){
91          perror("bind");
92          exit(EXIT_FAILURE);
93      }
94
95      /* Se sono arrivato qui significa che le operazioni precedenti sono andate a
96         buon fine e posso liberare l'area di memoria della lista res           */
97
98      freeaddrinfo(res);
99
100     if (listen(sd, SOMAXCONN) < 0)
101     {
102         perror("listen");
103         exit(EXIT_FAILURE);
104     }
105
106     /* Attendo i client */
107     for (;;){
108         //Estraggo una richiesta dalla coda e associo il descrittore della socket
            attiva ad ns
109         if ((ns = accept(sd, NULL, NULL)) < 0){
110             perror("accept");
111             exit(EXIT_FAILURE);
112         }
113
114         if ((pid == fork()) < 0){
115             perror("fork");
116             exit(EXIT_FAILURE);
117         }else if (pid == 0){
118             rxb_t rxb;
119             char categoria[MAX_REQUEST_SIZE];
120             char produttore[MAX_REQUEST_SIZE];
121             char ordinamento[MAX_REQUEST_SIZE]; //crescente o decrescente
122             size_t categoria_len, produttore_len, ordinamento_len;
123
124             close(sd);
125
126             /* Creo buffer rxb */
127             rxb_init(&rxb, MAX_REQUEST_SIZE * 2);
128
129             /* Disinstallo gestore segnali */
130             signal(SIGCHLD, SIG_DFL);
131
132             /* CIclo di richieste del singolo client */
133             for(;;){
134                 char *end_request = "\n--- END REQUEST ---\n";
135                 int pid_grep, pid_sort, status;
136                 int pipe_gs[2]; //Pipe tra il processo nipote grep e processo nipote
                    sort
137
```

```
138                 /* Leggere le tre informazioni */
139                 memset(categoria, 0, sizeof(categoria));
140                 categoria_len = sizeof(categoria) -1;
141                 if(rxb_readline(&rxb, ns, categoria, &categoria_len) < 0){
142                     perror("rxb_readline");
143                     exit(EXIT_FAILURE);
144                 }
145
146                 memset(produttore, 0, sizeof(produttore));
147                 produttore_len = sizeof(produttore) -1;
148                 if(rxb_readline(&rxb, ns, produttore, &produttore_len) < 0){
149                     perror("rxb_readline");
150                     exit(EXIT_FAILURE);
151                 }
152
153                 memset(ordinamento, 0, sizeof(ordinamento));
154                 ordinamento_len = sizeof(ordinamento) -1;
155                 if(rxb_readline(&rxb, ns, ordinamento, &ordinamento_len) < 0){
156                     perror("rxb_readline");
157                     exit(EXIT_FAILURE);
158                 }
159
160                 if(pipe(pipe_gs) < 0){
161                     perror("pipe");
162                     exit(EXIT_FAILURE);
163                 }
164
165                 /* Creo nipote che si occuperà della grep */
166                 if((pid_grep = fork()) < 0){
167                     perror("fork");
168                     exit(EXIT_FAILURE);
169                 }else if(pid_grep == 0){ //NIPOTE che si occuperà della grep
170
171                     char filename[MAX_REQUEST_SIZE + 256];
172
173                     //chiudo ciò che non serve
174                     close(pipe_gs[0]);
175                     close(ns);
176                     /* Reindirizzamento stdout su pipe */
177                     close(1); //stdout
178                     if(dup(pipe_gs[1]) < 0){
179                         perror("dup");
180                         exit(EXIT_FAILURE);
181                     }
182                     close(pipe_gs[1]);
183
184                     // snprintf(filename, sizeof(filename),
                        "/var/local/last_minute_gifts/%s.txt", categoria);
185                     snprintf(filename, sizeof(filename), "last_minute_gifts/%s.txt",
                        categoria);
186
187                     execlp("grep", "grep", produttore, filename, (char *)NULL);
188                     perror("exec grep");
189                     exit(EXIT_FAILURE);
190                 }
191
192                 /* Creo nipote che si occuperà del sort */
193                 if((pid_sort = fork()) < 0){
194                     perror("fork");
195                     exit(EXIT_FAILURE);
196                 }else if(pid_sort == 0){
197
198                     close(pipe_gs[1]);
199
200                     /* Reindirizzamento pipe su stdin */
201                     close(0); //stdout
202                     if(dup(pipe_gs[0]) < 0){
203                         perror("dup");
204                         exit(EXIT_FAILURE);
205                     }
206                     close(pipe_gs[0]);
```

```
207
208                     /* Reindirizzamento stdout su socket ns */
209                     close(1); //stdout
210                     if(dup(ns) < 0){
211                         perror("dup");
212                         exit(EXIT_FAILURE);
213                     }
214                     close(ns);
215
216                     if(strcmp(ordinamento, "decrescente") == 0){
217                         execlp("sort", "sort", "-n", "-r", (char *)NULL);
218                         perror("exec sort");
219                         exit(EXIT_FAILURE);
220                     }else{
221                         execlp("sort", "sort", "-n", (char *)NULL);
222                         perror("exec sort");
223                         exit(EXIT_FAILURE);
224                     }
225                 }
226
227                 /* chiudo la pipe non usata nel figlio */
228                 close(pipe_gs[0]);
229                 close(pipe_gs[1]);
230
231                 /* aspetto terminazione nipoti */
232                 waitpid(pid_grep, &status, 0);
233                 waitpid(pid_sort, &status, 0);
234
235                 /* mando la stringa di fine richiesta */
236                 write_all(ns, end_request, strlen(end_request));
237             }
238             // cleanup
239             close(ns);
240             rxb_destroy(&rxb);
241             exit(EXIT_SUCCESS);
242         }
243         // PADRE
244         close(ns);
245     }
246     close(sd);
247
248     return 0;
249 }
```

```c
// Client C esame 2021-12-20

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

#define MAX_REQUEST_SIZE 4096

int main(int argc, char **argv)
{
    int err;
    struct addrinfo hints, *res, *ptr;
    char *host_remoto;
    char *servizio_remoto;
    int sd, i = 1;
    rxb_t rxb;

    /* Controllo argomenti client <host> <porta> */
    if (argc != 3)
    {
        puts("Argomenti non validi! Uso: ./client host> <port>");
        exit(EXIT_FAILURE);
    }

    // Costruzione dell'indirizzo
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC;     //Compatibile con IPv4 e IPv6
    hints.ai_socktype = SOCK_STREAM; //Connessione affidabile con TCP

    // RIsoluzione dell'host
    host_remoto = argv[1];
    servizio_remoto = argv[2];

    if ((err = getaddrinfo(host_remoto, servizio_remoto, &hints, &res)) != 0)
    {
        fprintf(stderr, "Errore risoluzione nome: %s\n", gai_strerror(err));
        exit(EXIT_FAILURE);
    }

    for (ptr = res; ptr != NULL; ptr = ptr->ai_next)
    {

        // se socket fallisce salto direttamente alla prossima interazione
        if ((sd = socket(ptr->ai_family, ptr->ai_socktype, ptr->ai_protocol)) < 0)
        {
            fprintf(stderr, "connessione socket fallita\n");
            continue;
        }

        // se connect funziona allora esco dal ciclo for
        if (connect(sd, ptr->ai_addr, ptr->ai_addrlen) == 0)
        {
            printf("connect riuscita al tentativo %d\n", i);
            break;
        }
        i++;
        close(sd);
    }

    // verficia sul risultato restituito da getaddrinfo
```

```
72          if (ptr == NULL)
73          {
74              fprintf(stderr, "Errore risoluzione nome: nessun indirizzo corrispondente
                trovato\n");
75              exit(EXIT_FAILURE);
76          }
77
78          /* Liberiamo la memoria allocata da getaddrinfo() */
79          freeaddrinfo(res);
80
81          /* Inizializzo buffer di ricezione */
82          rxb_init(&rxb, MAX_REQUEST_SIZE);
83
84          for (;;)
85          {
86              char richiesta[MAX_REQUEST_SIZE];
87
88              //leggo da tastiera la categoria di prodotto
89              printf("\nInserisci il nome della categotia di regali: ");
90              if (fgets(richiesta, sizeof(richiesta), stdin) == NULL)
91              {
92                  perror("fgets");
93                  exit(EXIT_FAILURE);
94              }
95
96              /* Esco se l'utente digita . */
97              if (strcmp(richiesta, "fine\n") == 0)
98              {
99                  exit(EXIT_SUCCESS);
100             }
101
102             /* Invio richiesta al Server */
103             if (write_all(sd, richiesta, strlen(richiesta)) < 0)
104             {
105                 perror("write");
106                 exit(EXIT_FAILURE);
107             }
108
109             //leggo da tastiera il nome del produttore
110             printf("\nInserisci il nome del produttore: ");
111             if (fgets(richiesta, sizeof(richiesta), stdin) == NULL)
112             {
113                 perror("fgets");
114                 exit(EXIT_FAILURE);
115             }
116
117             /* Esco se l'utente digita . */
118             if (strcmp(richiesta, "fine\n") == 0)
119             {
120                 exit(EXIT_SUCCESS);
121             }
122
123             /* Invio richiesta al Server */
124             if (write_all(sd, richiesta, strlen(richiesta)) < 0)
125             {
126                 perror("write");
127                 exit(EXIT_FAILURE);
128             }
129
130             //leggo da tastiera il tipo di ordinamento
131             printf("\nInserire il tipo di ordinamento in base al prezzo desiderato
                (crescente o decrescente): ");
132             if (fgets(richiesta, sizeof(richiesta), stdin) == NULL)
133             {
134                 perror("fgets");
135                 exit(EXIT_FAILURE);
136             }
137
138             /* Esco se l'utente digita . */
139             if (strcmp(richiesta, "fine\n") == 0)
140             {
```

```
141             exit(EXIT_SUCCESS);
142         }
143 
144         /* Invio richiesta al Server */
145         if (write_all(sd, richiesta, strlen(richiesta)) < 0)
146         {
147             perror("write");
148             exit(EXIT_FAILURE);
149         }
150 
151         /* Leggo la risposta del server e la stampo a video */
152         for (;;)
153         {
154             char risposta[MAX_REQUEST_SIZE];
155             size_t risposta_len;
156 
157             memset(risposta, 0, sizeof(risposta));
158             risposta_len = sizeof(risposta) - 1;
159 
160             if (rxb_readline(&rxb, sd, risposta, &risposta_len) < 0)
161             {
162                 //Se sono qui è perchè ho ricevuto un EOF
163                 rxb_destroy(&rxb);
164                 fprintf(stderr, "Connessione chiusa dal server!\n");
165                 exit(EXIT_FAILURE);
166             }
167 
168             puts(risposta); //stampo a video la linea della risposta
169 
170             /* Passo a nuova richiesta una volta terminato input Server */
171             if (strcmp(risposta, "--- END REQUEST ---") == 0)
172             {
173                 break;
174             }
175         }
176     }
177 
178     //Chiudo la socket
179     close(sd);
180 
181     return 0;
182 }
```

```java
// Client java esame 2021-12-20

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

public class RegaliClient{
    public static void main(String[] args) {

        if(args.length != 2){
            System.err.println("Argomenti non validi! Utilizzo corretto: java RegaliClient <host> <port>");
            System.exit(1);
        }

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

            for(;;){
                //leggo la categoria da tastiera
                System.out.println("Inserire la cateogira di regali: ");
                String categoria = userIn.readLine();

                /* Esco se l'utente digita 'fine' */
                if (categoria.equals("fine")) {
                    break;
                }

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

                //leggo il nome del produttore da tastiera
                System.out.println("Inserire il nome del produttore: ");
                String produttore = userIn.readLine();

                /* Esco se l'utente digita 'fine' */
                if (produttore.equals("fine")) {
                    break;
                }

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


                //leggo il tipo di ordinamento da tastiera
                System.out.println("Inserire il tipo di ordinamento in base al prezzo desiderato (crescente o decrescente): ");
                String ordinamento = userIn.readLine();

                /* Esco se l'utente digita 'fine' */
                if (ordinamento.equals("fine")) {
                    break;
                }

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

                /* Leggo la risposta del server e la stampo a video */
                String risposta;
                for(;;){
                    // Leggo una riga alla volta
                    risposta = networkIn.readLine();
```

```
67                      /* Controllo errori */
68                      if (risposta == null) {
69                          System.err.println("Errore! Il Server ha chiuso la
                            connessione!");
70                          System.exit(2);
71                      }
72
73                      System.out.println(risposta);
74
75                      /* Passo a nuova richiesta una volta terminato input Server */
76                      if (risposta.equals("--- END REQUEST ---")) {
77                          break;
78                      }
79                  }
80              }
81
82              //Chiudo la socket
83              theSocket.close();
84
85          }catch (IOException e) {
86              System.err.println(e.getMessage());
87              e.printStackTrace();
88              System.exit(2);
89          }
90      }
91  }
```