```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <errno.h>
#include <fcntl.h>
#include <dirent.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define DIM_BUFF 100
#define LENGTH_NAME 20
#define max(a,b) ((a) > (b) ? (a) : (b))
#define N 10

/*************************************************************/

typedef struct
{
    /* Tipo di lunghezza massima LENGTH_NAME */
    char tipo[LENGTH_NAME];
    /* Luogo di lunghezza massima LENGTH_NAME */
    char luogo[LENGTH_NAME];
}
Request;


/*************************************************************/
typedef struct
{
    /* Descrizione di lunghezza massima LENGTH_NAME */
    char descrizione[LENGTH_NAME];
    /* Tipo di lunghezza massima LENGTH_NAME */
    char tipo[LENGTH_NAME];
    /* Luogo di lunghezza massima LENGTH_NAME */
    char luogo[LENGTH_NAME];
    /* disponibilita*/
    int disponibilita;
    /* Prezzo */
    int prezzo;
    /* Data*/
    int data[3];

}
Evento;

/*************************************************************/

// Definizione funzione per gestire il segnale SIGCHLD
void gestore(int signo) {
    int stato;
    printf("esecuzione gestore di SIGCHLD\n");
    wait(&stato);
}

/*************************************************************/
```

```
void visualizza(Evento elenco[]){
    int i;
    printf("\nDESCRIZIONE \t|| TIPO \t|| DATA\t\t|| LUOGO \t|| DISP \t||
        PREZZO\n");
    for(i=0; i<N; i++){
        printf("%s \t||", elenco[i].descrizione);
        printf("%s \t||", elenco[i].tipo);
        printf("%d/%d/%d \t||", elenco[i].data[0], elenco[i].data[1], elenco[i
            ].data[2]);
        printf("%s \t||", elenco[i].luogo);
        printf("%d \t\t||", elenco[i].disponibilita);
        printf("%d", elenco[i].prezzo);
        printf("\n");
    }

}
/*************************************************************/

int main(int argc, char * argv[]) {

    int listenfd, connfd, udpfd, nready, maxfdp1, nread, nwrite;
    const int on = 1;
    char buff[6*LENGTH_NAME+6], buff_udp[N*(6*LENGTH_NAME+6)];
    char luogo[LENGTH_NAME], tipo[LENGTH_NAME];
    fd_set rset;
    int i, j, len, port, eventoLength, prezzo;
    struct sockaddr_in cliaddr, servaddr;
    Evento elenco[N];
    Request req;

    /* CONTROLLO ARGOMENTI ------------------------------------------ */
    if (argc != 2)
    { printf("Error: %s port\n", argv[0]); exit(1); }
    // controllo per verificare che il numero di porta passato come
    // parametro sia un intero
    nread = 0;
    while (argv[1][nread] != '\0') {
        if (argv[1][nread] < '0' || argv[1][nread] > '9')
        { printf("Secondo argomento non intero\n"); exit(2);}
        nread++;
    }
    port = atoi(argv[1]);
    if (port < 1024 || port > 65535)
    { printf("Porta non valida"); exit(3); }

    printf("\nSelect_Server avviato\n");

    /* INIZIALIZZAZIONE STRUTTURA DATI ---------------------------------- */
    for(i=0; i<N; i++){
        strcpy(elenco[i].descrizione, "L");
        strcpy(elenco[i].tipo, "L\t");
        elenco[i].data[0]=-1;
        elenco[i].data[1]=-1;
        elenco[i].data[2]=-1;
        strcpy(elenco[i].luogo, "L\t");
        elenco[i].disponibilita=-1;
        elenco[i].prezzo=-1;
    }
    strcpy(elenco[0].descrizione, "String");
    strcpy(elenco[0].tipo, "Concerto");
```

```
elenco[0].data[0]=11;
elenco[0].data[1]=1;
elenco[0].data[2]=2014;
strcpy(elenco[0].luogo, "Verona");
elenco[0].disponibilita=40;
elenco[0].prezzo=40;
strcpy(elenco[2].descrizione, "Junentus-Inger");
strcpy(elenco[2].tipo, "Calcio");
elenco[2].data[0]=3;
elenco[2].data[1]=5;
elenco[2].data[2]=2014;
strcpy(elenco[2].luogo, "Torino");
elenco[2].disponibilita=21;
elenco[2].prezzo=150;
strcpy(elenco[4].descrizione, "GP Bologna");
strcpy(elenco[4].tipo, "Formula1");
elenco[4].data[0]=7;
elenco[4].data[1]=9;
elenco[4].data[2]=2014;
strcpy(elenco[4].luogo, "Bologna");
elenco[4].disponibilita=10;
elenco[4].prezzo=200;
visualizza(elenco);
printf("\n");

/* CREAZIONE SOCKET TCP ------------------------------------------------
    */
listenfd = socket(AF_INET, SOCK_STREAM, 0);
if (listenfd < 0)
{ perror("apertura socket TCP "); exit(1);}
printf("Creata la socket TCP d'ascolto, fd=%d\n", listenfd);

// Inizializzazione indirizzo server
memset((char *) &servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = INADDR_ANY;
servaddr.sin_port = htons(port);

// Set opzioni socket
if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on)) < 0)
{ perror("set opzioni socket TCP"); exit(2);}
printf("Set opzioni socket TCP ok\n");

// Bind socket TCP
if (bind(listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0)
{ perror("bind socket TCP"); exit(3); }
printf("Bind socket TCP ok\n");

// Coda d'ascolto per il server
if (listen(listenfd, 5) < 0)
{ perror("listen"); exit(4);}
printf("Listen ok\n");

/* CREAZIONE SOCKET UDP ------------------------------------------------
    */
udpfd = socket(AF_INET, SOCK_DGRAM, 0);
if (udpfd < 0)
{ perror("apertura socket UDP"); exit(5);}
printf("Creata la socket UDP, fd=%d\n", udpfd);
```

```
// Inizializzazione indirizzo server
memset((char *) &servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = INADDR_ANY;
servaddr.sin_port = htons(port);

// Set opzioni socket
if (setsockopt(udpfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on)) < 0)
{ perror("set opzioni socket UDP"); exit(6); }
printf("Set opzioni socket UDP ok\n");

// Bind socket UDP
if (bind(udpfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0)
{ perror("bind socket UDP"); exit(7); }
printf("Bind socket UDP ok\n");

/* Aggancio gestore per evitare figli zombie ----------------------------
    */
signal(SIGCHLD, gestore);

/* PULIZIA E SETTAGGIO MASCHERA DEI FILE DESCRIPTOR --------------------
    */
FD_ZERO(&rset);
maxfdp1 = max(listenfd,udpfd) + 1;

/* CICLO DI RICEZIONE EVENTI DELLA SELECT ------------------------------
    */
for (;;) {
    // Includo nella maschera rset le posizioni di listenfd e udpfd a uno
    FD_SET(listenfd, &rset);
    FD_SET(udpfd, &rset);

    if ((nready = select(maxfdp1, &rset, NULL, NULL, NULL)) < 0) {
        if (errno == EINTR) continue;
        else { perror("select"); exit(8); }
    }

    /* GESTIONE DELLE RICHIESTE ---------------------------------- */

    // Gestione richieste da socket STREAM
    if (FD_ISSET(listenfd, &rset)) {

        printf("Ricevuta richiesta per visualizzare eventi di un tipo in
            un luogo\n");

        // Lunghezze indirizzo client
        len = sizeof(struct sockaddr_in);

        // Creo una nuova connessione con il client -> nuova socket
        if ((connfd = accept(listenfd, (struct sockaddr *) &cliaddr, &len)
            ) < 0)
            { if (errno == EINTR) continue;
            else { perror("accept"); exit(9);}
        }
        // Generazione processo figlio
        if (fork() == 0) {

            // Chiudo la socket d'ascolto non necessaria in questa fase
            close(listenfd);
            printf("Dentro il figlio, pid=%i\n", getpid());
```

```
                // Viene generato un figlio per ogni richiesta

                nwrite = 0;
                // Accetto richieste fino alla fine del file
                while ((nread = read(connfd, &req, sizeof(Request))) > 0) {

                    strcpy(tipo,req.tipo);
                    strcpy(luogo,req.luogo);
                    printf("Ricevuta richiesta per eventi di tipo %s nel luogo
                        %s. \n", tipo, luogo);


                    for (i = 0; i < N; i++) {
                        if ((strcmp(elenco[i].descrizione, "L") != 0) &&
                            (strcmp(elenco[i].tipo, tipo) == 0) &&
                            (strcmp(elenco[i].luogo, luogo) == 0)) {

                            // Invio evento
                            // Preparazione stringa di risposta
                            strcpy(buff, elenco[i].descrizione);
                            strcat(buff, " \t||");
                            strcat(buff, elenco[i].tipo);
                            strcat(buff, " \t||");
                            sprintf(buff, "%s%d", buff, elenco[i].data[0]);
                            strcat(buff, "/");
                            sprintf(buff, "%s%d", buff, elenco[i].data[1]);
                            strcat(buff, "/");
                            sprintf(buff, "%s%d", buff, elenco[i].data[2]);
                            strcat(buff, " \t||");
                            strcat(buff, elenco[i].luogo);
                            strcat(buff, " \t||");
                            sprintf(buff, "%s %d", buff, elenco[i].
                                disponibilita);
                            strcat(buff, " \t\t||");
                            sprintf(buff, "%s %d", buff, elenco[i].prezzo);
                            strcat(buff, "\n");

                            // Invio lunghezza della stringa contenente
                                l'evento.
                            eventoLength = strlen(buff)+1;
                            eventoLength = htonl(eventoLength);
                            if ((nwrite = write(connfd, &eventoLength, sizeof
                                (int))) < 0) {
                                perror("write");
                                break;
                            }
                            // Invio della stringa.
                            if ((nwrite = write(connfd, buff, strlen(buff)+1))
                                < 0) {
                                perror("write");
                                break;
                            }

                        }
                    }
                    // Invio lunghezza zero (0) per segnalare la fine della
                        lista.
                    eventoLength = 0;
                    eventoLength = htonl(eventoLength);
```

```
                if ((nwrite = write(connfd, &eventoLength, sizeof(int))) <
                    0) {
                    perror("write");
                    break;
                }

                printf("Terminato invio al cliente\n");

            } // while

            printf("Figlio %i: chiudo connessione e termino\n", getpid());

            // Libero le risorse non piu' utilizzate
            close(connfd);
            exit(0);

        } // figlio

    // Padre chiude la socket dell'operazione
    close(connfd);

    } // Fine richieste stream

    // GESTIONE RICHIESTE DA SOCKET DATAGRAM
    if (FD_ISSET(udpfd, &rset)) {

        len=sizeof(struct sockaddr_in);
        if (recvfrom(udpfd, &prezzo, sizeof(int), 0, (struct sockaddr*)&
            cliaddr, &len)<0)
            {perror("recvfrom"); continue;}

        printf("Ricevuta richiesta per visualizzare eventi con prezzo
            inferiore a %d\n", prezzo);

        // Processa richiesta
        strcpy(buff_udp, "");
        j=0;
        for (i = 0; i < N; i++) {
            if (strcmp(elenco[i].descrizione, "L") != 0 && elenco[i].
                prezzo <= prezzo) {
                strcat(buff_udp, elenco[i].descrizione);
                strcat(buff_udp, " \t||");
                strcat(buff_udp, elenco[i].tipo);
                strcat(buff_udp, " \t||");
                sprintf(buff_udp, "%s%d", buff_udp, elenco[i].data[0]);
                strcat(buff_udp, "/");
                sprintf(buff_udp, "%s%d", buff_udp, elenco[i].data[1]);
                strcat(buff_udp, "/");
                sprintf(buff_udp, "%s%d", buff_udp, elenco[i].data[2]);
                strcat(buff_udp, " \t||");
                strcat(buff_udp, elenco[i].luogo);
                strcat(buff_udp, " \t||");
                sprintf(buff_udp, "%s%d", buff_udp, elenco[i].
                    disponibilita);
                strcat(buff_udp, " \t\t||");
                sprintf(buff_udp, "%s%d", buff_udp, elenco[i].prezzo);
                strcat(buff_udp, "\n");
                j++;
            }
        }
```

```
                if (j==0) strcpy(buff_udp, "N");

                // Invio risposta al cliente
                if (sendto(udpfd, buff_udp, sizeof(buff_udp), 0, (struct sockaddr
                    *)&cliaddr, len)<0)
                        {perror("sendto"); break;}


            } // Fine gestione Datagram

        } // Ciclo for della select
        exit(0);
    }
```