Lezione 16

- *Si completi l'esempio del maiuscolatore, scrivendo il codice del client. La struttura di quest'ultimo sarà la seguente:*

```
#include <ctype.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

main() {
  int sockfd;

  if( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("chiamata alla system call socket fallita");
    exit(1);
  }
  /* connessione al server */
  /* invio e ricezione della stringa */
  /* chiusura della connessione */
}
```

Un possibile programma client che acquisisca una stringa da standard input per trasmetterla al server e riceverla trasformata in maiuscolo è il seguente:

```
#include <ctype.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define MAXLENGTH 80
#define SERVER_PORT 1313

main() {
  int sockfd;
  struct sockaddr_in server={AF_INET,htons(SERVER_PORT),INADDR_ANY};
  int i=0,len;
  char buf[MAXLENGTH],c;

  if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("chiamata alla system call socket fallita");
    exit(1);
  }

  /* connessione al server */
  if(connect(sockfd,(struct sockaddr *)&server,sizeof server)==-1) {
```

```
    perror("connessione al server fallita");
    exit(2);
  }

  /* ricezione e stampa a video del messaggio di benvenuto del server */
  if(recv(sockfd,buf,27,0)>0) {
      buf[27]='\0';
      printf("%s",buf);
  }
  else {
      perror("Connessione al server interrotta");
      exit(3);
  }

  /* acquisizione della stringa da standard input */
  while((c=getchar())!='\n' && i<MAXLENGTH)
      buf[i++]=c;

  buf[i]='\0';
  len=strlen(buf);

  /* invio e ricezione della stringa */
  if(send(sockfd,buf,len,0)==-1) {
      perror("Errore nell'invio della stringa al server");
      exit(4);
  }

  if(recv(sockfd,buf,len,0)>0) {
      printf("%s\n",buf);
  }
  else {
      perror("Connessione al server interrotta");
      exit(3);
  }

  /* chiusura della connessione */
  close(sockfd);
}
```

- *Modificare il programma* ***upperserver.c*** *in modo che accetti più connessioni contemporaneamente (utilizzando la* ***fork****).*

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
```

```
#define SERVER_PORT 1313
#define LINESIZE 80

void upperlines(int in, int out) {
  char inputline[LINESIZE];
  int  len, i;

  while ((len = recv(in, inputline, LINESIZE, 0)) > 0) {
    for (i=0; i < len; i++)
      inputline[i] = toupper(inputline[i]);
    send(out, inputline, len, 0);
  }
}

int main (unsigned argc, char **argv) {
  int sock, client_len, fd;
  struct sockaddr_in server, client;

  /* impostazione del transport end point */
  if((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("chiamata alla system call socket fallita");
    exit(1);
  }

  server.sin_family = AF_INET;
  server.sin_addr.s_addr = INADDR_ANY;
  server.sin_port = htons(SERVER_PORT);

  /* binding dell'indirizzo al transport end point */
  if (bind(sock, (struct sockaddr *)&server, sizeof server) == -1) {
    perror("chiamata alla system call bind fallita");
    exit(2);
  }

  /* impostiamo il server in modo che possa gestire 5 richieste
   * contemporaneamente
   */
  listen(sock, 5);

  /* gestione delle connessioni dei client */
  while (1) {
    client_len = sizeof(client);
    if ((fd = accept(sock, (struct sockaddr *)&client, &client_len)) < 0) {
      perror("accepting connection");
      exit(3);
    }

    /* ogni volta che il server accetta una nuova connessione,
     * quest'ultima viene gestita da un nuovo processo figlio
     */
```

```
    switch(fork()) {
        case -1:
            perror("Errore nella chiamata alla fork");
            exit(4);
        case 0:
            fprintf(stderr, "Aperta connessione (PID %d).\n",getpid());
            send(fd, "Benvenuto all'UpperServer!\n", 27, 0);
            upperlines(fd, fd);
            close(fd);
            fprintf(stderr, "Chiusa connessione (PID %d).\n",getpid());
            exit(0);
        default :
            /* elimina eventuali figli zombie */
            while(waitpid(-1, 0, WNOHANG)>0);
    }

  }
}
```