4° Esercitazione (svolta):

Socket C senza connessione

Sviluppare un'applicazione C/S che consente di effettuare le quattro **operazioni tra interi in remoto**:

- il client invia al server pacchetti contenenti il tipo di operazione richiesta (somma, sottrazione, moltiplicazione o divisione) e gli operandi su cui effettuarla (due interi), che sono richiesti dall'utente usando l'input da console. Poi stampa la risposta, cioè il risultato dell'operazione;
- il server estrae i dati della richiesta, esegue l'operazione, e invia al client un datagramma contenente il risultato.

Si noti inoltre che client e server devono gestire opportunamente eventuali **fallimenti** delle operazioni invocate, si veda a tale proposito (anche consultando il **man** di Linux) a cosa serve la funzione **`perror`**.

Schema di soluzione: il client

Inizializzazione indirizzo del client (vedere meglio anche il codice più avanti) e del server, utilizzando gli argomenti di invocazione:

```
// inizializzazione servaddr
memset((char *)&servaddr, 0,
   sizeof(struct sockaddr_in));
servaddr.sin_family = AF_INET;
host = gethostbyname (argv[1]);
servaddr.sin_addr.s_addr =
   ((struct in_addr *)(host->h_addr))->s_addr;
servaddr.sin_port = htons(atoi(argv[2]);
```

Creazione e binding socket datagram:

```
sd=socket(AF_INET, SOCK_DGRAM, 0);
bind(sd,(struct sockaddr *) &clientaddr,
   sizeof(clientaddr));
```

Lettura da console dei dati della richiesta (operandi e operatore) …

Invio richiesta operazione al server:

```
sendto(sd, req, sizeof(Request), 0,
            (struct sockaddr *)&servaddr, len);
```

Ricezione risposta contenente il risultato dal server:

```
recvfrom(sd, &ris, sizeof(ris), 0,
           (struct sockaddr *)&servaddr, &len);
```

Chiusura socket e terminazione:

```
close(sd);
```


Schema di soluzione: il server

Inizializzazione indirizzo:

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

Creazione, bind, e settaggio opzioni della socket:

```
sd=socket(AF_INET, SOCK_DGRAM, 0);
bind(sd,(struct sockaddr *) &servaddr,
   sizeof(servaddr));
setsockopt(sd, SOL_SOCKET, SO_REUSEADDR,
   &on, sizeof(on));
```

Ricezione della richiesta dal client:

```
recvfrom(sd, req, sizeof(Request), 0,
   (struct sockaddr *)&cliaddr, &len);
```

Calcolo del risultato …

Invio della risposta al client:

```
sendto(sd, &ris, sizeof(ris), 0,
   (struct sockaddr *)&cliaddr, len);
```

E la chiusura della socket?

OpDatagram Client

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#define LINE_LENGTH 256
/**************************************/
typedef struct
{
  int op1;
  int op2;
  char tipoOp;
}
Request;
/**************************************/
int main(int argc, char **argv)
{
  struct hostent *host;
  struct sockaddr_in clientaddr, servaddr;
  int  port, sd, num1, num2, len, ris;
  char ok[LINE_LENGTH];
  char c;

  Request req;

  /* CONTROLLO ARGOMENTI ------------ */
  if(argc!=3)
  {
    printf("Error:%s serverAddress serverPort\n",
    argv[0]);
    exit(1);
  }
```



```
/* INIZIALIZZAZIONE INDIRIZZO CLIENT E SERVER */
memset((char *)&clientaddr, 0,
                sizeof(struct sockaddr_in));
clientaddr.sin_family = AF_INET;
clientaddr.sin_addr.s_addr = INADDR_ANY;
clientaddr.sin_port = 0;
memset((char *)&servaddr, 0,
       sizeof(struct sockaddr_in));
servaddr.sin_family = AF_INET;
host = gethostbyname (argv[1]);
/* VERIFICA INTERO */
num1=0;
while( argv[2][num1]!= '\0' ){
  if( (argv[2][num1] < '0') ||
      (argv[2][num1] > '9') )
  {
    printf("Secondo argomento non intero\n");
    exit(2);
  }
  num1++;
}
port = atoi(argv[2]);
if (host == NULL)
{
  printf("%s not found in /etc/hosts\n",
                  argv[1]);
  exit(2);
}
else
{
  servaddr.sin_addr.s_addr=
    ((struct in_addr *)(host->h_addr))->s_addr;
  servaddr.sin_port = htons(port);
}

/* CREAZIONE SOCKET ----------- */
sd=socket(AF_INET, SOCK_DGRAM, 0);
if(sd<0) {perror("apertura socket"); exit(1);}
printf("Client: creata la socket sd=%d\n", sd);

/* BIND SOCKET, a una porta scelta dal sistema*/
if(bind(sd,(struct sockaddr *) &clientaddr,
            sizeof(clientaddr))<0)
{perror("bind socket "); exit(1);}
```

```
/* CORPO DEL CLIENT: ciclo di acc. richieste */
printf("Qualsiasi tasto per procedere, EOF per \
  terminare: ");
/* ATTENZIONE!!
 * Cosa accade se la riga e' piu' lunga di
 * LINE_LENGTH-1?
 * Stesso dicasi per le altre gets...
 * Come si potrebbe risolvere il problema?
 */
while (gets(ok))
{

  /* lettura dei dati da inviare */
  printf("Primo operando (intero): ");
  while (scanf("%i", &num1) != 1)
  {
    /* Problema nell'implementazione della
     * scanf. Se l'input contiene PRIMA
     * dell'intero altri caratteri la testina
     * di lettura si blocca sul primo carattere
     * (non intero) letto. Ad esempio:
     * |ab1292\n|
     *  ^      La testina si blocca qui
     *  |
     * Bisogna quindi consumare tutto il buffer
     * in modo da sbloccare la testina.
     */
    do
    {c=getchar(); printf("%c ", c);}
    while (c!= '\n');
    printf("Primo operando (intero): ");
  }
  req.op1=htonl(num1);
  gets(ok);
  printf("Stringa letta: %s\n", ok);
  printf("Secondo operando (intero): ");
  while (scanf("%i", &num2) != 1)
  {
    do
    {c=getchar(); printf("%c ", c);}
    while (c!= '\n');
    printf("Secondo operando (intero): ");
  }
```



```
req.op2=htonl(num2);
gets(ok);
printf("Stringa letta: %s\n", ok);    do
{
  printf("Operazione (+ = addizione, - = \
    sottrazione, * = moltiplicazione, / = \
    divisione): ");
  c = getchar();
}
while (c!='+' && c !='-' && c!='*' &&
       c  !='/');
req.tipoOp=c;
gets(ok);
printf("Stringa letta: %s\n", ok);
/* lettura completata */
printf("Operazione richiesta: %ld %c %ld \n",
  ntohl(req.op1), req.tipoOp,
  ntohl(req.op2));

/* richiesta operazione */
len=sizeof(servaddr);
if(sendto(sd, &req, sizeof(Request), 0,
     (struct sockaddr *)&servaddr, len)<0)
{
  perror("sendto");
  continue;
}
```

```
    /* ricezione del risultato */
    printf("Attesa del risultato...\n");
    if (recvfrom(sd, &ris, sizeof(ris), 0,
          (struct sockaddr *)&servaddr, &len)<0)
    {perror("recvfrom"); continue;}

    printf("Esito dell'operazione: %i\n",
                                ntohl(ris));

    printf("Qualsiasi tasto per procedere, EOF \
      per terminare: ");
  }

  /* CHIUSURA SOCKET */
  close(sd);
  printf("\nClient: termino...\n");
  exit(0);
}
```


OpDatagram Server

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>

/****************************************/
typedef struct
{
  int op1;
  int op2;
  char tipoOp;
}
Request;
/****************************************/

int main(int argc, char **argv)
{
  int  sd;
  int port, len, num1, num2, ris;
  const int on = 1;
  struct sockaddr_in cliaddr, servaddr;
  struct hostent *clienthost;
  Request* req =(Request*)malloc(sizeof(Request));
  /* CONTROLLO ARGOMENTI ------------------- */
  if(argc!=2)
  {
    printf("Error: %s port\n", argv[0]);
    exit(1);
  }
```

```
else{
  /* VERIFICA INTERO----- */
  ...
  port = atoi(argv[1]);
}

/* 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);
/* CREAZ., BIND E SET. OPZIONI SOCKET --*/
sd=socket(AF_INET, SOCK_DGRAM, 0);
if(sd <0)
{perror("creazione socket "); exit(1);}
printf("Server: creata la socket, sd=%d\n", sd);
if(bind(sd,(struct sockaddr *) &servaddr,
                        sizeof(servaddr))<0)
{perror("bind socket "); exit(1);}
printf("Server: bind socket ok\n");
if(setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &on,
                         sizeof(on))<0)
{perror("set opzioni socket "); exit(1);}
printf("Server: set opzioni socket ok\n");

/* CICLO DI RICEZIONE RICHIESTE -------- */
for(;;)
{
  len=sizeof(struct sockaddr_in);
  if (recvfrom(sd, req, sizeof(Request), 0,
        (struct sockaddr *)&cliaddr, &len)<0)
  {perror("recvfrom "); continue;}
```



```
  /* trattiamo le conversioni possibili */
  num1=ntohl(req->op1);
  num2=ntohl(req->op2);
  printf("Operazione richiesta: %i %c %i\n",
     num1, req->tipoOp, num2);
  clienthost=gethostbyaddr( (char *)
    &cliaddr.sin_addr, sizeof(cliaddr.sin_addr),
    AF_INET);
  if (clienthost == NULL) printf(
    "client host information not found\n");
  else printf(
    "Operazione richiesta da: %s %i\n",
    clienthost->h_name,
    (unsigned)ntohl(cliaddr.sin_port));

  if(req->tipoOp=='+')
    ris=num1+num2;
  else if(req->tipoOp=='-')
    ris=num1-num2;
  else if(req->tipoOp=='*')
    ris=num1*num2;
  else if(req->tipoOp=='/')
    if (num2!=0)  ris=num1/num2;
  /* Risultato di default, in caso di errore.
     Sarebbe piu' corretto avere messaggi di
     errore, farlo per esercizio ------ */
  else ris=0;
  ris=htonl(ris);
  if (sendto(sd, &ris, sizeof(ris), 0,
        (struct sockaddr *)&cliaddr, len)<0)
  {perror("sendto "); continue;}

 }

}
```

Socket C con connessione

Sviluppare un'applicazione C/S che effettui l'**ordinamento remoto di un file** inviato dal client al server e restituito ordinato al client stesso, che lo scrive nel proprio file system:

- il client chiede all'utente il nome del file da ordinare e del file ordinato; se il file da ordinare è presente, viene stampato a video, inviato via uno stream di output al server e ricevuto ordinato dal server stesso, per poi essere salvato con il nome del file ordinato e stampato a video.

- il server attende una richiesta di connessione da parte dei client. Usa la connessione per ricevere il file, e inviare, se disponibile, il file ordinato.
  L'ordinamento locale viene fatto invocando il comando "sort" e usando la ridirezione: l'input e l'output del comando, anziché lo standard input e lo standard output, vengono ridiretti sulla socket.
  Il server deve gestire le richieste in maniera concorrente.

Schema di soluzione: il client

Inizializzazione indirizzo del server dall'argomento di invocazione:

```
memset((char *)&servaddr, 0,
   sizeof(struct sockaddr_in));
servaddr.sin_family = AF_INET;
host = gethostbyname (argv[1]);
servaddr.sin_addr.s_addr =
   ((struct in_addr *)(host->h_addr))->s_addr;
servaddr.sin_port = htons(port);
```

Creazione socket stream e connessione:

```
socket(AF_INET, SOCK_STREAM, 0);
connect(sd,(struct sockaddr *) &servaddr,
   sizeof(struct sockaddr));
```

Lettura da console dei dati della richiesta:
nome file da ordinare e nome file ordinato

Lettura e invio del file da ordinare al server:

```
while((nread=read(fd_sorg, buff, DIM_BUFF))>0){
   write(sd,buff,nread);}
```

Ricezione e scrittura del file ordinato dal server:

```
while((nread=read(sd, buff, DIM_BUFF))>0){
   write(fd_dest,buff,nread);}
```

Chiusura socket e terminazione:

```
close(sd);
```


Schema di soluzione: il server

Inizializzazione indirizzo:

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

Creazione, bind, settaggio opzioni, e creazione coda d'ascolto della socket:

```
socket(AF_INET, SOCK_STREAM, 0);
bind(listen_sd,(struct sockaddr *) &servaddr,
   sizeof(servaddr));
setsockopt(listen_sd, SOL_SOCKET, SO_REUSEADDR,
   &on, sizeof(on));
...
listen(listen_sd, 5);
```

Ricezione della richiesta dal client:

```
conn_sd = accept(listen_sd,
   (struct sockaddr *)&cliaddr,&len);
```

Nel processo **figlio**: ridirezione dei file descriptor di standard input e standard output, ed esecuzione comando di ordinamento (**`sort`**, vedere il man):

```
close(1); close(0);
dup(conn_sd); dup(conn_sd); close(conn_sd);
execl("/bin/sort", "sort", (char *)0);
```

Nel processo **padre**: chiusura della socket di servizio (non di ascolto!)

```
close(conn_sd);
```

RemoteSort Client

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

#define DIM_BUFF 256
#define LINE_LENGTH 256

int main(int argc, char *argv[])
{
  int sd, fd_sorg, fd_dest, nread;
  char buff[DIM_BUFF];
  char nome_sorg[FILENAME_MAX],
       nome_dest[FILENAME_MAX];
  struct hostent *host;
  struct sockaddr_in servaddr;

  /* CONTROLLO ARGOMENTI ---------------- */
  if(argc!=3)
  {
    printf("Error:%s serverAddress serverPort\n",
      argv[0]);
    exit(1);
  }
```



```
/* INIZIALIZZAZIONE INDIRIZZO SERVER ---- */
memset((char *)&servaddr, 0,
  sizeof(struct sockaddr_in));
servaddr.sin_family = AF_INET;
host = gethostbyname(argv[1]);
if (host == NULL)
{
  printf("%s not found in /etc/hosts\n",
    argv[1]);
  exit(1);
}

/* VERIFICA INTERO----- */
...
servaddr.sin_addr.s_addr=
  ((struct in_addr*) (host->h_addr))->s_addr;
servaddr.sin_port = htons(atoi(argv[2]));

/* CORPO DEL CLIENT: ------- */
printf("Ciclo di richieste di ordinamento fino \
  a EOF\n");
printf("Nome del file da ordinare, EOF per \
  terminare: ");

/* Utilizziamo la gets per consumare una
 * stringa immessa da tastiera (stdin).
 * Cosa accade se la stringa immessa e' piu'
 * lunga della struttura dati creata per
 * ospitarla (nome_sorg)? Vedere il man!!
 */
while (gets(nome_sorg))
{
  /* DEBUG: visualizzo il nome del file da
   * ordinare per verificare che non ci siano
   * errori
   */
  printf("File da aprire: __%s__\n", nome_sorg);
```

```
/* Verifico che il file da ordinare esista */
if((fd_sorg=open(nome_sorg, O_RDONLY))<0)
{
  perror("open");
  printf("Qualsiasi tasto per procedere, EOF \
    per terminare: ");
  continue;
}

printf("Nome del file ordinato: ");
if (gets(nome_dest)==0) break;

/* Creazione file ordinato */
if((fd_dest=open(nome_dest, O_WRONLY|O_CREAT,
                            0644))<0)
{
  perror("open");
  printf("Qualsiasi tasto per procedere, EOF \
                    per terminare: ");
  continue;
}

/* CREAZIONE SOCKET ------------- */
sd=socket(AF_INET, SOCK_STREAM, 0);
if(sd<0) {perror("apertura socket"); exit(1);}
printf("Client: creata la socket sd=%d\n",
  sd);
/* BIND implicita nella connect */
if(connect(sd,(struct sockaddr *) &servaddr,
       sizeof(struct sockaddr))<0)
{ perror("connect"); exit(1);}
printf("Client: connect ok\n");
```



```
    /* Invio e ricezione file ------------- */
    while((nread=read(fd_sorg, buff, DIM_BUFF))>0)
    {
      write(1,buff,nread);  // stampa a console
      write(sd,buff,nread);  // invio
    }
    shutdown(sd,1);
    while((nread=read(sd,buff,DIM_BUFF))>0)
    {
      write(fd_dest,buff,nread);
      write(1,buff,nread);
    }
    shutdown(sd, 0);
    close(fd_sorg); close(fd_dest);
    printf("Nome del file da ordinare, EOF per \
      terminare: ");
  }
  exit(0);
}
```

RemoteSort Server

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

/*************************/
void gestore(int signo)
{
  int stato;
  printf("esecuzione gestore di SIGCHLD\n");
  wait(&stato);
}
/*************************/

int main(int argc, char **argv)
{
  int  listen_sd, conn_sd;
  int port, len;
  const int on = 1;
  struct sockaddr_in cliaddr, servaddr;
  struct hostent *host;
```



```
/* CONTROLLO ARGOMENTI ---------- */
if(argc!=2)
{
  printf("Error: %s port\n", argv[0]);
  exit(1);
}
else port = atoi(argv[1]);

/* VERIFICA INTERO----- */
...
/* 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);

/* CREAZIONE, BIND, SETTAGGIO SOCKET D'ASCOLTO*/
listen_sd=socket(AF_INET, SOCK_STREAM, 0);
if(listen_sd <0)
{perror("creazione socket "); exit(1);}
printf("Server: creata la socket d'ascolto per\
  le richieste di ordinamento, fd=%d\n",
  listen_sd);
if(bind(listen_sd,(struct sockaddr *) &servaddr,
                  sizeof(servaddr))<0)
{perror("bind socket d'ascolto"); exit(1);}
printf("Server: bind socket d'ascolto ok\n");
if(setsockopt(listen_sd, SOL_SOCKET,
  SO_REUSEADDR, &on, sizeof(on))<0)
{perror("..."); exit(1);}
printf("Server: set ok\n");
/* CREAZIONE CODA D'ASCOLTO */
if (listen(listen_sd, 5)<0)
{perror("listen"); exit(1);}
printf("Server: listen ok\n");
```

```
/* AGGANCIO GESTORE PER EVITARE FIGLI ZOMBIE,
 * Quali altre primitive potrei usare? E'
 * portabile su tutti i sistemi? Pregi/Difetti?
 */
signal(SIGCHLD, gestore);

/* CICLO DI RICEZIONE RICHIESTE ------ */
for(;;)
{
  if((conn_sd=accept(listen_sd,
     (struct sockaddr *)&cliaddr,&len))<0)
  {
    /* La accept puo' essere interrotta dai
     * segnali inviati dai figli alla loro
     * teminazione. Tale situazione va gestita
     * opportunamente. Vedere nel man a cosa
     * corrisponde la costante EINTR!
     */
    if (errno==EINTR)
    {
      perror("Forzo la continuazione della \
        accept");
      continue;
    }
    else exit(1);
  }
  /* FIGLIO */
  if (fork()==0)
  {
    /* Chiusura file descriptor non utilizzati
     * e ridirezione di stdin e stdout
     */
    close(listen_sd);
    close(1); close(0);
    dup(conn_sd); dup(conn_sd); close(conn_sd);
    /* Esecuzione ordinamento */
    execl("/bin/sort", "sort", (char *)0);
  }
```



```
    /* PADRE: chiusura socket (NON di ascolto) */
    close(conn_sd);
  }
}
```