Esercizio

- Scrivete un applicazione client-server per l’invio di messaggi usando la socket library.
- Il client invia messaggi in forma di stringhe, il server riceve le stringhe e le visualizza a schermo

Parte iniziale del client

- #include <stdio.h>
- #include <sys/types.h>
- #include <sys/socket.h>
- #include <netinet/in.h>
- #include <netdb.h>
- void error(char *msg){   perror(msg);   exit(0);}
- int main(int argc, char *argv[])
- {   int sockfd, portno, n;   struct sockaddr_in serv_addr;   struct hostent *server;   char buffer[256];
- if (argc < 3) {   fprintf(stderr,"usage %s hostname port\n", argv[0]);   exit(0);   }
- portno = atoi(argv[2]);
- sockfd = socket(AF_INET, SOCK_STREAM, 0);   if (sockfd < 0)   error("ERROR opening socket");

Client (ctd.)

- server = gethostbyname(argv[1]);
- if (server == NULL) {fprintf(stderr,"ERROR, no such host\n"); exit(0);}
- bzero((char *) &serv_addr, sizeof(serv_addr)); serv_addr.sin_family = AF_INET;
- bcopy((char *)server->h_addr,
- (char *)&serv_addr.sin_addr.s_addr, server->h_length);
- serv_addr.sin_port = htons(portno);
- if (connect(sockfd,&serv_addr,sizeof(serv_addr)) < 0) error("ERROR connecting");
- printf("Please enter the message: ");
- bzero(buffer,256);
- fgets(buffer,255,stdin);
- n = write(sockfd,buffer,strlen(buffer));
- if (n < 0) error("ERROR writing to socket");
- bzero(buffer,256); n = read(sockfd,buffer,255);
- if (n < 0) error("ERROR reading from socket"); printf("%s\n",buffer); return 0;}

Server

- * A simple server in the internet domain using TCP   The port number is passed as an argument */
- #include <stdio.h>
- #include <sys/types.h>
- #include <sys/socket.h>
- #include <netinet/in.h>
- void error(char *msg){   perror(msg);   exit(1);}
- int main(int argc, char *argv[]){   int sockfd, newsockfd, portno, clilen; char buffer[256];   struct sockaddr_in serv_addr, cli_addr; int n;

Server (ctd.)

- if (argc < 2) {   fprintf(stderr,"ERROR, no port provided\n");   exit(1);   }
- sockfd = socket(AF_INET, SOCK_STREAM, 0);
- if (sockfd < 0)   error("ERROR opening socket"); bzero((char *) &serv_addr, sizeof(serv_addr));
- portno = atoi(argv[1]);   serv_addr.sin_family = AF_INET;   serv_addr.sin_addr.s_addr = INADDR_ANY;   serv_addr.sin_port = htons(portno);

Server (fin)

- if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) error("ERROR on binding");
- listen(sockfd,5);
- clilen = sizeof(cli_addr);
- newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
- if (newsockfd < 0) error("ERROR on accept");   bzero(buffer,256);
- n = read(newsockfd,buffer,255);
- if (n < 0) error("ERROR reading from socket");
- printf("Here is the message: %s\n",buffer);
- n = write(newsockfd,"I got your message",18);
- if (n < 0) error("ERROR writing to socket");   return 0; }

Esame HTTP header

- http://www.softswot.com/http-header.php#viewer

Esempi RPC da http://www.cs.rpi.edu/~hollingd/netprog/code/rpc/

- simple: remote integer add and subtract procedures.
- ulookup: Remote username and uid lookup procedures. The server returns the username corresponding to a UID (on the server system) or the UID corresponding to a username. Among other things, this example show how to use the XDR string type.
- better_ulookup: Better remote username and uid lookup procedures. Almost the same as lookup, but now the procedures return complex types that include a status indicator (to indicate when the lookup failed).
- linkedlist: Example of how to use a linked list as an argument to a remote procedure.
- varray: Example that shows using a variable length array.
- rpctalk: RPC based talk program. This is an example of a hand-made RPC program and includes a custom dispatcher and asynchronous remote procedures. There is a single program that acts as both client and server. This code is old and is set up (the Makefile) to build on a Sun using the ucbsoc library (Sun RPC is now based on TLI). To run you need to add /usr/ucblib to your LD_LIBRARY_PATH environment variable (I can't find a static library...). Good luck.
- **There are also some RFCs here:**
  - RFC 1831 - Remote Procedure Call Protocol Specification Version 2
  - RFC 1832 - XDR: External Data Representation Standard

RPCtalk.x

- const MAXSTRING = 100;
- typedef stringmessage<MAXSTRING>;
- /* progid is a structure passed to SENDPROG */
- struct progid {string hostname<100>;
- /* the host name */
- u_long prognum;
- /* the RPC program number */};
- program TALK {version TALKVERS {   void SENDPROG(progid) = 1; void SENDMESG(message) = 2;} = 1;} = 222222;

RPCtalk.c

- #define PORTMAP
- #include <stdio.h>
- #include <sys/types.h>
- #include <sys/socket.h>
- #include <rpc/rpc.h>
- #include <string.h>
- #include "rpctalk.h"/* generated by the protocol compiler */

RPCtalk.c (ctd.)

- void setup_client(char *serverhost, u_long prog, u_long progvers);
- void eprint(char *s){ printf("%s\n",s); exit(1);}
- /* some global variables */
- SVCXPRT *serv_handle;
- /* transport handle used by the server */
- CLIENT *clnt_handle;/* client handle used when calling an RPC proc */
- int serv_sd;/* server socket - used by select() */
- int client_established;
- /* flag used to prevent multiple clients

RPCtalk.c

- /* define the remote procedures */
- /* procedure #1 gets a program number and hostname */
- void *sendprog_1(progid *p){if (client_established) {printf("Multiple talk partner rejected\n");   } else {client_established=1;
- setup_client(p->hostname,p->prognum,TALKVERS);
- printf("Talking with %s at prog %d\n",p->hostname,p->prognum);   } return(NULL);}

RPCtalk.c

- /* procedure #2 gets a message and prints it */
- void *sendmesg_2(message *s){   printf("Received: %s",*s); return(NULL);}
- /* RPC dispatcher - called by svc_getreqset(&rdset) */
- void dispatch(struct svc_req *rqstp, SVCXPRT *transp){ progid pg; char mesg[200]; char *mptr = mesg; switch (rqstp->rq_proc) {
- /* 0 is the null proc (RPC "ping") */ case 0:
- (void)svc_sendreply(transp, xdr_void, (char *)NULL);   break;
- case SENDPROG:
- /* other talk program is sending us it's RPC address */
- if (svc_getargs(transp,xdr_progid,(char *) &pg)==FALSE) { svcerr_decode(transp); return;}   sendprog_1(&pg);   break; case SENDMESG:
- /* other talk program is sending us a message */
- if (svc_getargs(transp,xdr_message,(char *)&mptr)==FALSE) { svcerr_decode(transp); return;}   sendmesg_2(&mptr);   break;

  default: svcerr_noproc(transp);   break; }}

RPCtalk.c

- /* create a udp socket, a transport handle,   and register with the portmapper register each rpc routine with   the dispatcher*/
- int setup_server(u_long prog, u_long progver){int serv_sd; int serv_port; struct sockaddr_in serv_addr; int serv_len;
- if ((serv_sd=socket(PF_INET,SOCK_DGRAM,0))==0)   eprint("Error creating socket"); serv_addr.sin_family = AF_INET; serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
- serv_addr.sin_port = htons(0);
- serv_len = sizeof(serv_addr); if (bind(serv_sd, (struct sockaddr *) &serv_addr, serv_len)<0) eprint("Error binding\n"); if ((serv_handle = svcudp_create(serv_sd))==NULL) eprint("Error creating server handle");
- /* find out what port we are using */
- if (getsockname(serv_sd, (struct sockaddr *) &serv_addr, &serv_len)<0) eprint("Error getsockname\n");
- serv_port = serv_addr.sin_port;
- /* register with the portmapper and the dispatcher */
- pmap_unset(prog,progver);/* unset just in case */
- if (svc_register(serv_handle, prog,progver, dispatch, IPPROTO_UDP)==FALSE)

  eprint("Problem registring with portmapper\n");   return(serv_sd);}

RPCtalk.c

- /* create a client handle which will be used to make RPC calls to   the other talk program (used to send messages)*/
- void setup_client(char *serverhost, u_long prog, u_long progvers){ struct timeval tm = {0,0};
- if ((clnt_handle = clnt_create(serverhost, prog, progvers, "udp"))==NULL) eprint("Problem creating client handle\n");
- /* set up client_handle to timeout immediately !!! */
- clnt_control(clnt_handle,CLSET_RETRY_TIMEOUT,&tm); clnt_control(clnt_handle,CLSET_TIMEOUT,&tm);}
- /* make an RPC call to the other talk program's SENDPROG procedure   this identifies us with the other process - telling it our   hostname and RPC program number*/
- void sendprognum(char *host,u_long prognum){ int stat; struct timeval tm = {0,0 } ;
- /* timeout immediately */
- progid pd;   pd.hostname = host; pd.prognum=prognum;
- stat = clnt_call(clnt_handle,SENDPROG,   xdr_progid, (char *) &pd, xdr_void, NULL, tm);
- /* we expect a timeout error since we timeout right away */
- /* RPC server is not sending us a response anyway */
- if (stat!=RPC_TIMEDOUT) eprint("Problem with sending my address\n"); }

- /* make an RPC call to the other talk program's SENDMESG procedure   this sends a message we got from the keyboard to the other talk   program*/
- void sendmessage(char *s){ int stat; struct timeval tm = {0,0 } ;
- stat = clnt_call(clnt_handle,SENDMESG,   xdr_message,(char *) &s, xdr_void, NULL, tm);
- /* we expect a timeout error since we timeout right away */
- /* RPC server is not sending us a response anyway */
- if (stat!=RPC_TIMEDOUT)   eprint("Error calling SENDMESG"); }
- /* custom svc_run(). This replaces the standard svc_run() supplied as   part of the RPC library. We need to provide our own so that we can look   for input from the keyboard as well as RPC requests from the other talk program.   svc_fdset is a global variable provided by the RPC library.   svc_fdset is already initialized with all socket descriptors   corresponding to ports we are using to service RPC requests*/
- void svc_run( ){ fd_set rdset; char s[101]; int readlen;
- while (strncasecmp(s,"quit",4)!=0)   {rdset = svc_fdset;
- /* get a copy of the RPC socket read set */
- FD_SET(0,&rdset);
- /* add stdin to the set */
- select(100, &rdset, NULL, NULL, NULL);/* first check to see if there is anything from the keyboard */
- if (FD_ISSET(0,&rdset))   {
- /* handle stdin */
- if ((readlen=read(0,s,100))==0) {
- /* EOF - quit */
- strcpy(s,"quit");   continue; }   else {   if (client_established) { s[readlen]=0;
- /* terminate the string */
- /* send the string to the other program */ sendmessage(s);}   }   }
- FD_CLR(0,&rdset);
- /* get rid of stdin before   calling svc_getreqset */svc_getreqset(&rdset);
- /* calls the dispatcher for   each pending request */   }}

RPCtalk.c

- /*main routine   if there is a command line argument assume it is the name of   a host that is already running this program (acting as an RPC   server for program number TALK).   if there is no command line argument - we are starting first.   Setup to be an RPC server for program number TALK*/
- main(int argc, char **argv){ u_long myprognum; char s[100]; if (argc==1)   {
- /* we are the initial server - set to be an RPC server   program number TALK   */
- serv_sd = setup_server(TALK,TALKVERS);
- myprognum = TALK;
- /*printf("Established as primary server\n"); */   } else   {
- /* we are the secondary talk program   generate a random RPC program number,   become an RPC server using that number   send the primary server our RPC address */
- myprognum = 300000+rand()%1000;serv_sd = setup_server(myprognum,TALKVERS);setup_client(argv[1],TALK,TALKVERS);if (gethostname(s,90)<0)   eprint("Error determining my hostname"); sendprognum(s,myprognum);client_established=1;
- /* printf("Established as client\n"); */   } svc_run();
- /* look for RPC requests and for input from keyboard */ pmap_unset(myprognum,1);/* unregister with the portmapper */}