Programação C/S, diálogo iterativo com protocolo TCP

A. Título: Programando aplicações não concorrentes para diálogo unidirecional, com Sockets TCP

B. Objetivos: Os alunos deverão, ao final do laboratório, dominar o que está relacionado abaixo:

- ⇒ Aprender como se constroem aplicações simples em rede com sockets TCP
- ⇒ Conhecer as funções connect, accept, listen, close, recv e send para serviços orientados à conexão.

C. Roteiro: O conteúdo deste laboratório pode ser explorado, considerando as subdivisões apresentadas a seguir:

Parte I – Comunicação Cliente/Servidor no mesmo host:

1) Entre no seu diretório pessoal, em um host Linux e crie um sub-diretório com o nome **labtcp_iterativo**.

2) Copie os códigos tcpClient.c e tcpServer.c disponibilizados para o sub-diretório criado e compile-os utilizando o gcc (ex.: gcc tcpClient.c –o tcpClient). Em linhas gerais esses códigos realizam as funções abaixo. Procure identificar onde elas ocorrem no código.

tcpServer.c	tcpClient.c
i) Preenche a estrutura sockaddr_in (endServ) c/dados do servidor	i) Preenche uma estrutura sockaddr_in (ladoServ) com dados do servidor
ii) Cria um socket TCP	ii) Preenche uma estrutura sockaddr_in (ladoCli) com os dados do cliente
iii) Relaciona o socket criado com os dados do servidor (bind)	iii) Cria um socket TCP (sd)
iv) Escuta o socket aguardando solicitações de conexão	iv) Para cada parâmetro estabelece conexão com o servidor (connect)
v) Aceita a conexão, criando novo socket para atender o remoto	v) Inicia diálogo unidirecional com o servidor (os dados enviados são informados pelo usuário)
vi) Recebe e imprime mensagens do remoto na tela	vi) O diálogo se encerra assim que o parâmetro FIM é enviado.
vii) Encerra conexão.	

3) Você precisará de 2 consoles do sistema e os passos abaixo deverão ser seguidos na ordem em que estão apresentados:
   a) Na console 1, ative o processo tcpServer passando como parâmetros o IP e a porta de escuta (ex.: ./tcpServer 127.0.0.1 5200, onde 127.0.0.1 é o IP do servidor e 5200 é a porta de escuta desse servidor).
   b) Na console 2, ative o processo tcpClient, passando como parâmetros o endereço IP e a porta de escuta do servidor (ex.: tcpClient 127.0.0.1 5200 Alô, onde 127.0.0.1 é o IP do servidor e 5200 é a porta que o servidor está escutando).
   c) Faça testes de emissão de dados do cliente para o servidor e confira se estão chegando corretamente no destinatário. Alterne para a console na qual o servidor está rodando para perceber isso.

Parte II – Comunicação Cliente/Servidor em hosts diferentes:

4) Escolha um dos micros como servidor e ative o processo tcpServer, na console 1, passando as novas informações de porta e IP deste servidor. A seguir, ative o processo tcpClient no outro micro, passando IP e porta do servidor. OBS: não ativar o servidor na sua loopback, mas no endereço específico de sua interface. Teste também a conexão de rede (por exemplo, através de ping).

5) Confira se o fluxo das informações está ok entre cliente e servidor pela comutação entre as consoles 1 e 2.

```c
/* ****************************************/
/*Lab.Redes 2  tcpServer.c (não concorrente) */
/****************************************/
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h> /* close */
#define SUCESSO 0
#define ERRO     1
#define MAX_MSG    100

int main (int argc, char *argv[]) {
  int sd, newSd, cliLen, n;
  struct sockaddr_in endCli;  /* Informacoes do Cliente */
  struct sockaddr_in endServ; /* Informacoes do Servidor */
  char rcv_msg[MAX_MSG];

  if (argc<3) {
        printf("Digite IP e Porta para este servidor\n");
        exit(1); }
  /* Criando o socket */
  sd = socket(AF_INET, SOCK_STREAM, 0);
  if(sd<0) {
    printf("nao foi possivel abrir o socket");
    return ERRO; }

  /* Preenchendo dados sobre este servidor */
  endServ.sin_family      = AF_INET;
  endServ.sin_addr.s_addr  = inet_addr(argv[1]);
  endServ.sin_port        =  htons(atoi(argv[2]));
  /* Fazendo bind na porta do servidor */
  if(bind(sd, (struct sockaddr *) &endServ, sizeof(endServ))<0) {
    printf("nao foi possivel fazer o bind ");
    return ERRO; }
  listen(sd,5);
  cliLen = sizeof(endCli);
  printf("%s Servidor aguardando conexÃ£o ... ", argv[0]);
  while (1) {
    /* aceita a conexao do cliente */
    newSd = accept(sd, (struct sockaddr *) &endCli, &cliLen);
    if(newSd<0) {
        printf("nao foi possivel aceitar a conexao ");
        return ERRO; }
    /* inicia a variavel que vai receber os dados */
    printf("----------- Aceitando conexao ---------\n");
    memset(rcv_msg,0x0,MAX_MSG); /* init buffer */
    /* recebe os dados desse cliente */
    n = recv(newSd, rcv_msg, MAX_MSG, 0); /* espera por dados */
    if (n<0) {
       printf("nao pode receber os dados");
       return ERRO; }
    printf("{TCP, IP_S: %s, Porta_S: %u,", inet_ntoa(endServ.sin_addr),
             ntohs(endServ.sin_port));
    printf(" IP_C: %s, Porta_C: %u} => %s\n",
inet_ntoa(endCli.sin_addr),
             ntohs(endCli.sin_port), rcv_msg);
    printf("----------- Encerrando conexao --------\n\n");
    close(newSd);
   } /* fim do while (1) */
   return SUCESSO;
} /* fim do programa */
```

```c
/*******************************/
/* Lab.Redes 2  tcpClient.c     */
/*******************************/

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h> /* close */

#define MAX_MSG     100

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

  int sd, rc, i;
  struct sockaddr_in ladoCli;  /* dados do cliente */
  struct sockaddr_in ladoServ; /* dados do servidor */

  if(argc < 4) {
    printf("uso: %s <ip_serv> <porta_serv> <data1> ...  <dataN>\n",argv[0]);
    exit(1);}

  /* dados do servidor */
  ladoServ.sin_family      = AF_INET;
  ladoServ.sin_addr.s_addr = inet_addr(argv[1]);
  ladoServ.sin_port        = htons(atoi(argv[2]));

  /* dados do cliente */
  ladoCli.sin_family      = AF_INET;
  ladoCli.sin_addr.s_addr = htonl(INADDR_ANY);
  ladoCli.sin_port        = htons(0);
  for (i=3;i<argc;i++) {
    /* criando um socket */
    sd = socket(AF_INET, SOCK_STREAM, 0);
    if(sd<0) {
      printf("nao foi possivel abrir o socket ");
      exit(1);}

    /* faz um bind para a porta escolhida */
    rc = bind(sd, (struct sockaddr *) &ladoCli, sizeof(ladoCli));
    if(rc<0) {
      printf("nao foi possivel fazer o bind na porta TCP");
      exit(1); }

    printf("------ Estabelecendo conexao -------\n");
     /* faz a conexao com o servidor */
    rc = connect(sd, (struct sockaddr *) &ladoServ, sizeof(ladoServ));
    if(rc<0) {
       printf("nao foi possivel conectar");
       exit(1);}
    printf("Enviando dado %u => %s \n", i-2, argv[i]);
    rc = send(sd, argv[i], strlen(argv[i]) + 1, 0);
    if(rc<0) {
       printf("nao foi possivel enviar dados");
       close(sd);
       exit(1);}
    printf("------ Encerrando conexao ---------\n\n");
    close(sd);
  } /* fim-do-for */
return 0;
} /* fim do programa */
```