Cliente-Servidor usando Sockets

1 - Aplicação com *Sockets* UDP

- executar primeiro o **servidor** e depois o **cliente**;
- executar primeiro o **cliente** e depois o **servidor**.

2 - Aplicação com *Sockets* TCP

- executar primeiro o **servidor** e depois o **cliente**;
- executar primeiro o **cliente** e depois o **servidor**.

3 - Aplicação TCP *versus* UDP

- **vantagens e desvantagens**

1 - Aplicação usando Sockets UDP
Cliente UDP

```
/* Include Files */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>


main(int argc, char **argv)
{
  int s;
  unsigned short port;
  struct sockaddr_in server;
  char buf[32], msg[32];
```

1 - Aplicação usando Sockets UDP ... Cliente UDP

```
/* argv[1] = internet address; argv[2] = port of server */
if(argc != 3) {
  printf("Usage: %s <host address> <port> \n",argv[0]);
  exit(1);
}

/* convert the port from ascii to integer and then
** from host byte order to network byte order. */
port = htons(atoi(argv[2]));

/* Create a datagram socket in the internet domain
** and use the default protocol (UDP). */
if ((s = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
  perror("socket()");
  exit(1);
}
```


1 - Aplicação usando Sockets UDP ... Cliente UDP

```
/* Set up the server name */
server.sin_family = AF_INET; /* Internet Domain */
server.sin_port = port;     /* Server Port */
/* Server's Address */
server.sin_addr.s_addr = inet_addr(argv[1]);

/* Put a message into the buffer */
printf("\n\nEntre com a mensagem: ");

if (fgets(msg, 31, stdin) == NULL ) {
  fprintf(stderr, "fgets failed\n");
  exit(2);
}

strcpy(buf, msg); /* copy message to buffer */
```

1 - Aplicação usando Sockets UDP ... Cliente UDP

```
  /* Send the message in buf to the server */
  if( sendto(s, buf, (strlen(buf)+1), 0, (struct sockaddr
   *)&server, sizeof(server)) < 0 )  {
    perror("sendto()");
    exit(2);
  }

  close(s);          /* Deallocate the socket */
}
```


1 - Aplicação usando Sockets UDP ... Servidor UDP

```
/* Include Files */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

main(int argc, char **argv)
{

   int sockint,s, namelen, client_address_size;
   struct sockaddr_in client, server;
   char buf[32];
```

1 - Aplicação usando Sockets UDP ... Servidor UDP

```
/* Create a datagram socket in the internet domain
** and use the default protocol (UDP). */
if ((s = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
  perror("socket()");
  exit(1);
}


/* Bind my name to this socket so that clients on the
** network can send me messages. This allows the operating
** system to demultiplex messages and get them to the
** correct server.*/
```


1 - Aplicação usando Sockets UDP ... Servidor UDP

```
/* Set up the server name. The internet address is specified
** as the wildcard INADDR_ANY so that the server can get
** messages from any of the physical internet connections on
** this host. (Otherwise we would limit the server to
** messages from only one network interface) */

/* Server is in Internet Domain */
server.sin_family = AF_INET;
server.sin_port = 0; /* Use any available port */

/* Server's Internet Address */
server.sin_addr.s_addr = INADDR_ANY;
```

1 - Aplicação usando Sockets UDP ... Servidor UDP

```
if(bind(s, (struct sockaddr *)&server, sizeof(server)) < 0)
{
  perror("bind()");
  exit(2);
}

/* Find out what port was really assigned and print it */
namelen = sizeof(server);
if(getsockname(s,(struct sockaddr *)&server,&namelen) < 0)
{
  perror("getsockname()");
  exit(3);
}

printf("Port assigned is %d\n", ntohs(server.sin_port));
```


1 - Aplicação usando Sockets UDP ... Servidor UDP

```
/* Receive a message on socket s in buf  of maximum size 32
** from a client. Because the last two paramters are not
** null, the name of the client will be placed into the
** client data structure and the size of the client address
** will be placed into client_address_size. */

client_address_size = sizeof(client);

if(recvfrom(s, buf, sizeof(buf), 0, (struct sockaddr *)
 &client, &client_address_size) <0)
{
  perror("recvfrom()");
  exit(4);
}
```

1 - Aplicação usando Sockets UDP ... Servidor UDP

```
  /* Print the message and the name of the client. The domain
  ** should be the internet domain (AF_INET). The port is
  ** received in network byte order, so we translate it to
  ** host byte order before printing it. The internet address
  ** is received as 32 bits in network byte order so we use a
  ** utility that converts it to a string printed in dotted
  ** decimal format for readability. */
  printf("Received message: %sfrom domain %s port %d internet
    address %s\n", buf,
    (client.sin_family == AF_INET?"AF_INET":"UNKNOWN"),
    ntohs(client.sin_port),
    inet_ntoa(client.sin_addr));

  /* Deallocate the socket. */
   close(s);
}
```


2 - Aplicação usando Sockets TCP Servidor TCP

```
/* Include Files. */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

/* Server Main. */
main(int argc, char **argv)
{
  unsigned short port;        /* port server binds to  */
  char buf[12];    /* buffer for sending and receiving data */
  struct sockaddr_in client;  /* client address information */
  struct sockaddr_in server;  /* server address information */

  int s;          /* socket for accepting connections */
  int ns;         /* socket connected to client */
  int namelen;    /* length of client name */
```

2 - Aplicação usando Sockets TCP ... Servidor TCP

```
/* Check Arguments: (only one) the port number to bind to */
if (argc != 2) {
  fprintf(stderr, "Usage: %s port\n", argv[0]);
  exit(1);
}

/* First argument should be the port. */
port = (unsigned short) atoi(argv[1]);

/* Get a socket for accepting connections. */
if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
  perror("Socket()");
  exit(2);
}
```


2 - Aplicação usando Sockets TCP ... Servidor TCP

```
/* Bind the socket to the server address. */
server.sin_family = AF_INET;
server.sin_port   = htons(port);
server.sin_addr.s_addr = INADDR_ANY;
if(bind(s,(struct sockaddr *)&server, sizeof(server)) < 0) {
  perror("Bind()");
  exit(3);
}

/* Listen for Connections. Specify the backlog as 1. */
if (listen(s, 1) != 0) {
  perror("Listen()");
  exit(4);
}
```

2 - Aplicação usando Sockets TCP ... Servidor TCP

```
  /* Accept a Connection. */
  namelen = sizeof(client);
  if( (ns = accept(s,(struct sockaddr *)&client, &namelen))
      == -1) {
    perror("Accept()");
    exit(5);
  }

  /* Receive the message on the newly connected socket. */
  if (recv(ns, buf, sizeof(buf), 0) == -1)    {
    perror("Recv()");
    exit(6);
  }
```


2 - Aplicação usando Sockets TCP ... Servidor TCP

```
  /*  Print the received message. */
  printf("\n\nMensagem: %s", buf);

  /* Send the message back to the client. */
  if (send(ns, buf, sizeof(buf), 0) < 0) {
    perror("Send()");
    exit(7);
  }

  close(ns);  close(s);
  printf("\nServer ended successfully\n");
  exit(0);
}
```

2 - Aplicação usando Sockets TCP ... Cliente TCP

```
/* Include Files. */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>

/* Client Main. */
main(int argc, char **argv)
{
  unsigned short port;  /* port client will connect to */
  char buf[12], msg[12];  /* data buffer for sending */
  char msg[12];  /* data buffer for receiving  */
```


2 - Aplicação usando Sockets TCP ... Cliente TCP

```
  struct hostent *hostnm;  /* server host name information */
  struct sockaddr_in server;  /* server address */
  int s;  /* client socket  */


  /* Check Arguments Passed. Should be hostname and port.  */
  if (argc != 3)    {
    fprintf(stderr, "Usage: %s hostname port\n", argv[0]);
    exit(1);
  }
```

2 - Aplicação usando Sockets TCP ... Cliente TCP

```
/* The host name is the first argument.
** Get the server address. */
hostnm = gethostbyname(argv[1]);
if (hostnm == (struct hostent *) 0)    {
  fprintf(stderr, "Gethostbyname failed\n");
  exit(2);
}

/* The port is the second argument.  */
port = (unsigned short) atoi(argv[2]);
```


2 - Aplicação usando Sockets TCP ... Cliente TCP

```
/* Put a message into the buffer. */
printf("\n\nEntre com a mensagem: ");
if (fgets(msg, 11, stdin) == NULL)      {
  fprintf(stderr, "fgets failed\n");
  exit(2);
}
/* copy message to buffer */
strcpy(buf, msg);

/* Put the server information into the server structure.
** The port must be put into network byte order. */
server.sin_family = AF_INET;
server.sin_port = htons(port);
server.sin_addr.s_addr = *((unsigned long *)hostnm->h_addr);
```

2 - Aplicação usando Sockets TCP ... Cliente TCP

```
 /* Get a stream socket. */
 if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
   perror("Socket()");
   exit(3);
}

 /* Connect to the server.  */
 if( connect(s,(struct sockaddr *)&server, sizeof(server))
   < 0) {
   perror("Connect()");
   exit(4);
 }
```


2 - Aplicação usando Sockets TCP ... Cliente TCP

```
 /* Connect to the server.  */
 if( connect(s,(struct sockaddr *)&server, sizeof(server))
   < 0) {
   perror("Connect()");
   exit(4);
 }

 /* send buffer */
 if (send(s, buf, sizeof(buf), 0) < 0) {
  perror("Send()");
  exit(5);
 }
```

2 - Aplicação usando Sockets TCP ... Cliente TCP

```
  /* The server sends back the same message.
  ** Receive it into the buffer. */
  if (recv(s, buf, sizeof(buf), 0) < 0)  {
    perror("Recv()");
    exit(6);
  }
  /* Print the returned message. */
  printf("\n\nMensagem retornada: %s", buf);

  close(s); /* Close the socket. */
  printf("\nClient Ended Successfully\n");
  exit(0);
}
```


3 - Aplicação TCP versus UDP vantagens x desvantagens

- decorrentes das características dos protocolos
  - Transport Control Protocol
    - ... notas de aula referente a Arquitetura TCP/IP !
  - User Datagram Protocol
    - ... notas de aula referente a Arquitetura TCP/IP !