Atividade 1 - Pseudo código de um cliente e servidor, em TCP e UDP

1 Introdução

Nesta atividade iremos detalhar a implementação (em pseudo-código) de um cliente e de um servidor básicos tanto para o protocolo TCP/IP quando para o UDP/IP.

TCP

O protocolo TCP é um protocolo orientado a conexões, logo para que qualquer transmissão ocorra é necessário primeiro estabelecer uma conexão, além disso, uma vez estabelecida uma conexão a comunicação ocorre em um fluxo (stream).

Tipicamente cada cliente do servidor é tratado como um arquivo, tendo assim seu próprio descritor, é importante pensar em concorrência em um servidor TCP pois ambas as operações de escrita e leitura são tipicamente bloqueantes e ocorrem por descritor.

1.1 Cliente

O cliente TCP não precisa saber sua própria porta de origem, o sistema se encarrega de escolher uma disponível, ele pode ser implementado da seguinte maneira:

Algorithm 1 Cliente TCP

```
1: function
2:     socket ← createTCPSocket()                    ▷ Cria um socket
3:     data ← dataToSend
4:     connectSocketToServer(socket,server_address,server_port)
5:     Send(data,socket)
6:     Receive(buffer,socket)
7:     Print(buffer)
8:     closesocket
```

1.2 Servidor

O servidor TCP como dito antes é sensível a bloqueios, assim é necessário pensar em concorrência mesmo em aplicação de baixo volume.

Outra característica do servidor é que o mesmo deve ouvir por clientes em uma porta especifica, não se conectar diretamente a eles.

Algorithm 2 Servidor TCP

```
1:  function LISTEN
2:      socket ← createTCPSocket()                    ▷ Cria um socket
3:      bindSocket(socket,port)                       ▷ Liga o servidor a uma porta
4:      listen(socket)                                ▷ Avisa o sistema que vai receber conexões
5:      while True do
6:          client ← accept(socket)                   ▷ Recebe uma conexão
7:          PID ← Fork
8:          if PID ≠ 0 then
9:              accept(client)
10:         else
11:             close(client)
12:     close(socket)
13: function ACCEPT(client)
14:     Receive(client,&buffer)
15:     Send(client,buffer)
```

UDP

1.3 Cliente

O cliente UDP é bem similar ao TCP, porém não é necessário usar criar uma conexão, pois os dados não seguem em fluxo, mas em mensagens (datagrams)

```
1: function
2:     socket ← createUDPSocket()                    ▷ Cria um socket
3:     data ← dataToSend
4:     SendTo(data,socket,serverAddress,serverPort)
5:     ReceiveFrom(&buffer,socket,&serverAddress,&serverPort)
6:     Print(buffer)
7:     closesocket
```

1.4 Servidor

O servidor UDP é bastante interessante, por sem um protocolo sem conexão ele é naturalmente não bloqueante, o que reduz muito a necessidade de concorrência no servidor.

```
1: function
2:     socket ← createUDPSocket()                          ▷ Cria um socket
3:     while True do
4:         ReceiveFrom(&buffer,socket,&clientAddress,&clientPort)
5:         SendTo(data,socket,clientAddress,clientPort)
6:     closesocket
```

2 Código em C

O código abaixo implementa os dois servidores e clientes, uma linha em main decide qual das 4 aplicações deve ser compilada

```
/*************************************************************/
/* Gabriel Hidasy Rezende, RA116928 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include<arpa/inet.h>
#include<signal.h>
void err(char *msg)
{
  fprintf(stderr,"%s\n",msg);
  exit(0);
}

int stopsignal = 1;
void sighandler(int signo)
{
  if (signo == SIGINT) {
    printf("Server shutting down, this can take a minute...\n");
    stopsignal = 0;
  }
}
```

```
/* Exercicio 1a, um exemplo de codigo para um servidor TCP */

void TCPserverAccepter(int client)
{
  send(client,"200 OK\nHI\n",10,0);
  close(client);
}
void TCPserverListener()
{
  int sockfd, client, port = 12345;
  char buffer[1024];
  struct sockaddr_in serv_addr, cli_addr;
  sockfd = socket(AF_INET, SOCK_STREAM,0);
  if(sockfd < 0) {
    err("Erro abrindo o socket TCP de servidor");
  }
  //Ler sobre os campos dessa estrutura
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = INADDR_ANY;
  serv_addr.sin_port = htons(port);

  int len = sizeof(serv_addr);

  if(bind(sockfd, (struct sockaddr *) &serv_addr, len) < 0) {
    err("Erro no binding do servidor TCP");
  }
  listen(sockfd,5);

  int pid;
  while (stopsignal) {
    client = accept(sockfd, (struct sockaddr *) &cli_addr, &len);
    pid = fork();
    if (pid < 0) {
      close(client);
      err("Erro no fork()");
      continue;
    }
    if (pid == 0) {
      /*Trata o cliente*/
      TCPserverAccepter(client);
      break;
    }
    /*Loga o pid do filho pra matar o servidor de forma limpa com um C-c*/
  }
  close(sockfd);
  return;
```

```
}

/* Exercicio 1b, um exemplo de codigo para um cliente TCP */
void TCPClient() {
  struct sockaddr_in my_addr, server_addr;
  int sockfd;
  sockfd = socket(AF_INET,SOCK_STREAM,0);
  memset(&my_addr,0,sizeof(my_addr));
  my_addr.sin_family = AF_INET;
  my_addr.sin_port = htons(12345);
  my_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
  if (sockfd < 0) {
    err("Erro com o socket do cliente TCP");
  }
  int len = sizeof(my_addr);
  if(connect(sockfd, (struct sockaddr*) &my_addr, len) < 0) {
    err("Erro na conect do cliente TCP");
  }
  fprintf(stdout,"Cliente conectado!\n");
  char buffer[1024];
  recv(sockfd,&buffer,1024,0);
  fprintf(stdout,"from the server:\n%s",buffer);
  close(sockfd);
  return;
}

void UDPServer()
{
  int sockfd, client, port = 12345;
  char buffer[1024];
  struct sockaddr_in serv_addr, cli_addr;
  sockfd = socket(AF_INET, SOCK_DGRAM,0);
  if(sockfd < 0) {
    err("Erro abrindo o socket UDP de servidor");
  }
  //Ler sobre os campos dessa estrutura
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = INADDR_ANY;
  serv_addr.sin_port = htons(port);

  int len = sizeof(serv_addr);

  if(bind(sockfd, (struct sockaddr *) &serv_addr, len) < 0) {
    err("Erro no binding do servidor UDP");
  }
  /* O servidor UDP não usa listen(n) */
```

```
  int pid;
  int rlen;
  while (stopsignal) {
    /* Logo ele também não roda accept, só recv */
    rlen = recvfrom(sockfd, buffer, 1024, 0, (struct sockaddr *) &cli_addr, &len);
    buffer[rlen] = 0;
    printf("%s\n",buffer);
    sendto(sockfd,buffer,rlen,0,(struct sockaddr *) &cli_addr,sizeof(cli_addr));
    /* E como não há conexão, o socket não fica preso a um cliente, */
    /*o que diminue  muito a necessidade de forks ou threads */
  }
  close(sockfd);
  return;
}

/* Exercicio 2b, um exemplo de codigo para um cliente UDP */
void UDPClient() {
  /* UDP é um pouco diferente de TCP, aqui não há coneção*/
  struct sockaddr_in server_addr;
  int sockfd;
  /* A primeira diferença é que o socket é de datagram */
  sockfd = socket(AF_INET,SOCK_DGRAM,0);
  memset(&server_addr,0,sizeof(server_addr));
  server_addr.sin_family = AF_INET;
  server_addr.sin_port = htons(12345);
  server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
  if (sockfd < 0) {
    err("Erro com o socket do cliente UDP");
  }
  int len = sizeof(server_addr);
  char sbuffer[1024] = "mensagem do cliente pra você\nola\n";
  char rbuffer[1024];
  int rlen;
  sendto(sockfd,sbuffer,strlen(sbuffer),0,(struct sockaddr *) &server_addr, sizeof(server_ad
  rlen = recvfrom(sockfd,rbuffer,1024,0,NULL,NULL);
  rbuffer[rlen] = 0;
  fprintf(stdout,"from the server:\n%s",rbuffer);
  close(sockfd);
  return;
}

int main (int argc, char *argv[])
{
  signal(SIGINT,sighandler);
  //TCPserverListener();
```

```
  //TCPClient();
  //UDPClient();
  UDPServer();
}
```