Engenharia de Telecomunicações
STD29006 – Sistemas Distribuídos

Prof. Emerson Ribeiro de Mello
mello@ifsc.edu.br

Laboratório 2: Sockets TCP e UDP: Java e C

12/11/2021

Esse laboratório tem por objetivo apresentar como desenvolver aplicações cliente e servidor em Java e em C usando a API *sockets*.

1 Introdução a programação com sockets

- API C criada em 1983 no 4.2 BSD UNIX, é padrão em todos S.O. No windows é a `winsock`
- Socket consiste de um ponto final para comunicação bidirecional entre duas partes, cliente e servidor. O canal de comunicação trata-se na verdade de dois sockets interconectados
- Na comunicação cliente e servidor:
  - Servidor aguarda por conexão de um cliente
  - Servidor pode atender vários clientes (ex: fork, threads)
  - Baixo acoplamento: A única exigência é que cliente e servidor falem a mesma língua, ou seja, que exista um protocolo bem definido e conhecido por ambos. Desta forma, cliente e servidor podem ser até desenvolvidos em diferentes linguagens de programação ou serem executados em diferentes sistema operacionais ou arquiteturas de máquinas

2 Sockets em C

Em C, um socket é acessado através de um descritor, de forma análogo a um descritor de arquivos. Pode-se então executar operações de escrita (write) e leitura (read). Funções necessárias para trabalhar com sockets estão definidas em `<sys/sockets.h>`, sendo estas:

- `socket` – para criar um socket
- `bind` – para associar um nome ao socket
- `listen` – para ouvir em uma determinada porta
- `accept` – para receber conexões
- `connect` – para conectar
- `read` – para ler dados
- `write` – para escrever dados
- `close` – para fechar

Para desenvolver uma aplicação servidora com sockets TCP, deve-se: (1) criar um socket; (2) associar um nome ao socket (é aqui que deve-se indicar o IP e porta); (3) ouvir em uma determinada porta; (4) aceitar as conexões dos clientes; (5) realizar comunicação, por exemplo, recebendo uma mensagem do cliente e depois devolvendo a mesma como um eco; (6) fechar a conexão. Abaixo estes passos apresentados em detalhes.

2.1 Criando um socket

Ao criar um socket deve-se informar 3 parâmetros: domínio, tipo e protocolo.

Domínio

- `AF_UNIX` – somente para processos dentro de uma mesma máquina
- `AF_INET` – para processos em diferentes máquinas – IPv4
- `AF_INET6` – IPv6

Tipo

- `SOCK_STREAM` – orientado a conexão com transmissão de fluxos de bytes, sequencial e bidirecional
- `SOCK_DGRAM` – orientado a datagramas, com tamanho fixo e entrega não confiável
- `SOCK_RAW` – interface de datagrama diretamente para ir sobre o IP (camada de rede)

Protocolo – Geralmente se informa o valor 0 para que se escolha automaticamente o protocolo adequado para o domínio e tipo. Se um domínio e tipo puder fazer uso de diferentes protocolos, então deve-se indicar se deseja TCP ou UDP.

```c
/* Criando um socket */
// AF_INET = ARPA INTERNET PROTOCOLS -- IPv4
// SOCK_STREAM = orientado a conexao
// 0 = protocolo padrao para o tipo escolhido -- TCP
int socket_desc = socket(AF_INET , SOCK_STREAM , 0);

if (socket_desc == -1){
  printf("Nao foi possivel criar socket\n");
  return -1;
}
```

2.2 Associando um nome e ouvindo

```c
//Preparando a struct do socket
struct sockaddr_in servidor;
servidor.sin_family = AF_INET;
servidor.sin_addr.s_addr = INADDR_ANY; // Obtem IP do S.O.
servidor.sin_port = htons( 1234 );

//Associando o socket a porta e endereco
if( bind(socket_desc,(struct sockaddr *)&servidor , sizeof(servidor)) < 0){
  puts("Erro ao fazer bind\n");
}
puts("Bind efetuado com sucesso\n");

// Ouvindo por conexoes
listen(socket_desc, 3);
```

Arquiteturas diferentes usam diferentes formas para representar internamente a ordem dos bytes. Leia sobre representação de dados: *big-endian* e *little-endian*. As funções abaixo visam garantir que a ordem dos bytes nativa de cada arquitetura seja convertida para uma ordem de byte para a rede e vice-versa, garantindo assim harmonia na comunicação.

- **Regra de ouro:** Converta para o formato da rede antes de enviar (veja linha 15) e lembre de converter de volta para o formato nativo após receber algo pela rede (veja linha 39).
  - `htons` – host to network short;
  - `htonl` – host to network long;
  - `ntohs` – network to host short;
  - `ntohl` – network to host long.

2.3 Aceitando conexões

```
1  //Aceitando e tratando conexoes
2  struct sockaddr_in cliente;
3  int c;
4  puts("Aguardando por conexoes...");
5  c = sizeof(struct sockaddr_in);
6  conexao = accept(socket_desc, (struct sockaddr *)&cliente, (socklen_t*)&c);
7  if (conexao<0){
8    perror("Erro ao receber conexao\n");
9    return -1;
10 }
```

2.4 Realizando comunicação e fechando o socket

```
1  // pegando IP e porta do cliente
2  char *cliente_ip;
3  int cliente_port;
4  cliente_ip = inet_ntoa(cliente.sin_addr);
5  cliente_port = ntohs(cliente.sin_port);
6  printf("cliente conectou: %s : [%d]\n",cliente_ip, cliente_port);
7
8  // lendo dados enviados pelo cliente
9  if((tamanho = read(conexao,resposta, 1024)) < 0){
10   perror("Erro ao receber dados do cliente: ");
11   return -1;
12 }
13
14 // Coloca terminador de string
15 resposta[tamanho] = '\0';
16 printf("O cliente falou: %s\n", resposta);
17
18 // Enviando resposta para o cliente
19 mensagem = "Ola cliente, tudo bem?";
20 write(conexao , mensagem , strlen(mensagem));
21
22 // fechando
23 shutdown(socket_desc, 2);
24 printf("Servidor finalizado...\n");
```

2.5 Código completo em C de um servidor TCP

Baixe o código fonte clicando aqui.

```c
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

#define MAX_MSG 1024

/*
    Experimento 01

    Servidor aguarda por mensagem do cliente, imprime na tela
    e depois envia resposta e finaliza processo
*/

int main(void)
{
    //variaveis
    int socket_desc, conexao, c;
    struct sockaddr_in servidor, cliente;
    char *mensagem;
    char resposta[MAX_MSG];
    int tamanho, count;

    // para pegar o IP e porta do cliente
    char *cliente_ip;
    int cliente_port;

    /*******************************************************/
    //Criando um socket
    socket_desc = socket(AF_INET, SOCK_STREAM, 0);
    if (socket_desc == -1)
    {
        printf("Nao foi possivel criar o socket\n");
        return -1;
    }

    int reuso = 1;
    if (setsockopt(socket_desc, SOL_SOCKET, SO_REUSEADDR, (const char *)&reuso, sizeof(reuso)) < 0)
    {
        perror("Não foi possível reusar endereço");
        return -1;
    }
#ifdef SO_REUSEPORT
    if (setsockopt(socket_desc, SOL_SOCKET, SO_REUSEPORT, (const char *)&reuso, sizeof(reuso)) < 0)
    {
        perror("Não foi possível reusar porta");
        return -1;
    }
#endif

    //Preparando a struct do socket
    servidor.sin_family = AF_INET;
    servidor.sin_addr.s_addr = INADDR_ANY; // Obtem IP do S.O.
    servidor.sin_port = htons(1234);

    //Associando o socket a porta e endereco
    if (bind(socket_desc, (struct sockaddr *)&servidor, sizeof(servidor)) < 0)
    {
        perror("Erro ao fazer bind\n");
        return -1;
    }
    puts("Bind efetuado com sucesso\n");

    // Ouvindo por conexoes
    listen(socket_desc, 3);
    /*******************************************************/

    //Aceitando e tratando conexoes
    puts("Aguardando por conexoes...");
    c = sizeof(struct sockaddr_in);
    conexao = accept(socket_desc, (struct sockaddr *)&cliente, (socklen_t *)&c);
    if (conexao < 0)
    {
        perror("Erro ao receber conexao\n");
        return -1;
    }
    /*******************************************************/

    /*********comunicao entre cliente/servidor***************/

    // pegando IP e porta do cliente
    cliente_ip = inet_ntoa(cliente.sin_addr);
    cliente_port = ntohs(cliente.sin_port);
    printf("cliente conectou\nIP:PORTA -> %s:%d\n", cliente_ip, cliente_port);

    // lendo dados enviados pelo cliente
    if ((tamanho = read(conexao, resposta, MAX_MSG)) < 0)
    {
        perror("Erro ao receber dados do cliente: ");
```

```
        return -1;
    }

    // Coloca terminador de string
    resposta[tamanho] = '\0';
    printf("O cliente falou: %s\n", resposta);

    // Enviando resposta para o cliente
    mensagem = "Ola cliente, tudo bem?";
    write(conexao, mensagem, strlen(mensagem));

    /*******************************************************/

    // http://www.gnu.org/software/libc/manual/html_node/Closing-a-Socket.html
    shutdown(socket_desc, 2);

    printf("Servidor finalizado...\n");
    return 0;
}
```

2.6 Código completo em C de um cliente TCP

Baixe o código fonte clicando aqui.

```
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#define MAX_MSG 1024
/*
  Experimento 01

  Cliente envia mensagem ao servidor e imprime resposta
  recebida do Servidor

  Cliente conhece o endereco IPv4 do servidor
*/
int main(int argc, char *argv[])
{
  // variaveis
  int socket_desc;
  struct sockaddr_in servidor;
  char *mensagem;
  char resposta_servidor[MAX_MSG];
  int tamanho;

  /***************************************/
  /* Criando um socket */
  // AF_INET = ARPA INTERNET PROTOCOLS
  // SOCK_STREAM = orientado a conexao
  // 0 = protocolo padrao para o tipo escolhido -- TCP
  socket_desc = socket(AF_INET, SOCK_STREAM, 0);

  /* Informacoes para conectar no servidor */
  // IP do servidor
  // familia ARPANET
  // Porta - hton = host to network short (2bytes)
  servidor.sin_addr.s_addr = inet_addr("127.0.0.1");
  servidor.sin_family = AF_INET;
  servidor.sin_port = htons(1234);

  //Conectando no servidor remoto
  if (connect(socket_desc, (struct sockaddr *)&servidor, sizeof(servidor)) < 0)
  {
    printf("Nao foi possivel conectar\n");
    return -1;
  }
  printf("Conectado no servidor\n");
  /***************************************/

  /*******COMUNICAO - TROCA DE MENSAGENS **************/
  // Protocolo definido para essa aplicacao
  // 1 - Cliente conecta
  // 2 - Cliente envia mensagem
  // 3 - Servidor envia resposta

  //Enviando uma mensagem
  mensagem = "Oi servidor";
  if (send(socket_desc, mensagem, strlen(mensagem), 0) < 0)
  {
    printf("Erro ao enviar mensagem\n");
    return -1;
  }
  puts("Dados enviados\n");

  //Recebendo resposta do servidor
  if ((tamanho = recv(socket_desc, resposta_servidor, MAX_MSG, 0)) < 0)
  {
```

```
    printf("Falha ao receber resposta\n");
    return -1;
  }
  printf("Resposta recebida: ");
  resposta_servidor[tamanho] = '\0'; // adicionando fim de linha na string
  puts(resposta_servidor);
  /*****************************************/
  close(socket_desc); // fechando o socket
  printf("Cliente finalizado com sucesso!\n");
  return 0;
}
```

2.7 Outras versões de cliente e servidores TCP em C

- Baixe aqui um servidor TCP que se mantém ativo após atender um cliente.
- Baixe aqui um cliente TCP que conecta em um servidor HTTP por meio de nome
- Baixe aqui um servidor TCP que faz uso de *pthread*
  - Baixe aqui um cliente TCP para o servidor acima

3 Sockets TCP em Java

3.1 Código completo em Java de um servidor TCP

Baixe o código fonte clicando aqui.

```
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.net.ServerSocket;
import java.io.InputStreamReader;
import java.net.Socket;

public class ServidorTCP {

   public static void main(String[] args) {

      try (ServerSocket socket = new ServerSocket(1234)) {
         System.out.println("Aguardando por conexoes em: " +
                            socket.getInetAddress() + ":" + socket.getLocalPort());
         try (Socket clientSocket = socket.accept()) {
            // Estabelecendo fluxos de entrada e saída
           BufferedReader entrada = new BufferedReader(new InputStreamReader(clientSocket.
    getInputStream()));
           DataOutputStream saida = new DataOutputStream(clientSocket.getOutputStream());

            // lendo mensagem enviada pelo cliente
            String mensagem = entrada.readLine();
            System.out.println("Cliente> " + mensagem);

            // enviando mensagem para o cliente
            saida.writeBytes("Oi, eu sou o servidor!\n");

         } catch (Exception e) {
            System.err.println(e.toString());
         }
      } catch (Exception e) {
         System.err.println(e.toString());
      }
   }
}
```

3.2 Código completo em Java de um cliente TCP

Baixe o código fonte clicando aqui.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.DataOutputStream;
import java.net.Socket;

public class ClienteTCP {

    public static void main(String[] args) throws IOException{
        int servidorPorta = 1234;
        String servidorIP = "127.0.0.1";

        try(Socket conexao = new Socket(servidorIP, servidorPorta)){
            System.out.println("Conectado! " + conexao);

            /* Estabelece fluxos de entrada e saida */
            DataOutputStream saida = new DataOutputStream(conexao.getOutputStream());
            BufferedReader entrada = new BufferedReader(new InputStreamReader(conexao.
    getInputStream()));

            /* Inicia comunicacao */
            String mensagem = "Oi, eu sou o cliente!\n";
            saida.write(mensagem.getBytes());
            saida.flush();

            String recebida = entrada.readLine();
            System.out.println("Servidor> " + recebida);


        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }
}
```

4 Sockets UDP em C

4.1 Código completo em C de um servidor UDP

Baixe o código fonte clicando aqui.

```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/types.h>

#define TAMBUFFER 512
#define PORTA 1234

int main(void){
  struct sockaddr_in addr_local;
  struct sockaddr_in addr_remoto;
  int socket_desc;
  socklen_t slen = sizeof(addr_remoto);
  char buf[TAMBUFFER];
  int tam_recebido;

  // Criando um socket UDP
  if ((socket_desc=socket(AF_INET, SOCK_DGRAM, 0)) < 0 ){
    perror("nao foi possivel criar socket");
    return -1;
  }

  // ligando o socket ao IP e porta
  memset((char *) &addr_local, 0, sizeof(addr_local));
  addr_local.sin_family = AF_INET;
  addr_local.sin_addr.s_addr = htonl(INADDR_ANY);
  addr_local.sin_port = htons(PORTA);
  if (bind(socket_desc, (struct sockaddr *)&addr_local, sizeof(addr_local)) < 0){
    perror("erro ao fazer bind");
    return -1;
  }

  while(1){
    printf("Aguardando...\n");
    // processando pacote recebido
    if ( (tam_recebido = recvfrom(socket_desc, buf, TAMBUFFER, 0, (struct sockaddr *)&addr_remoto,
     &slen)) > 0){
      buf[tam_recebido]='\0';
      printf("Pacote recebido de: %s:[%d]\nDados: %s\n\n",
        inet_ntoa(addr_remoto.sin_addr), ntohs(addr_remoto.sin_port), buf);

      // respondendo ao addr_remoto
      if ((sendto(socket_desc, buf, strlen(buf), 0, (struct sockaddr *)&addr_remoto, slen)) <0){
        perror("erro ao enviar resposta");
      }
    }
  }

  close(socket_desc);
  return 0;
}
```

4.2 Código completo em C de um cliente UDP

Baixe o código fonte clicando aqui.

```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/types.h>

#define TAMBUFFER 512
#define PORTA 1234

int main(void){
  struct sockaddr_in addr_local;
  struct sockaddr_in addr_remoto;
  int socket_desc;
  socklen_t slen = sizeof(addr_remoto);
  char buf[TAMBUFFER];
  int tam_recebido;
  char *servidor = "127.0.0.1";

  // Criando um socket UDP
  if ((socket_desc=socket(AF_INET, SOCK_DGRAM, 0))==-1)
    printf("socket created\n");
  // Ligando o socket a todos IPs locais e escolhendo qualquer porta
  memset((char *)&addr_local, 0, sizeof(addr_local));
  addr_local.sin_family = AF_INET;
  addr_local.sin_addr.s_addr = htonl(INADDR_ANY);
  addr_local.sin_port = htons(0);
  if (bind(socket_desc, (struct sockaddr *)&addr_local, sizeof(addr_local)) < 0) {
    perror("erro ao fazer bind");
    return -1;
  }
  // Definindo addr_remoto como o endereco onde quero me conectar
  // Convertendo a string 127.0.0.1 para formato binario com inet_aton
  memset((char *) &addr_remoto, 0, sizeof(addr_remoto));
  addr_remoto.sin_family = AF_INET;
  addr_remoto.sin_port = htons(PORTA);
  if (inet_aton(servidor, &addr_remoto.sin_addr)==0) {
    fprintf(stderr, "inet_aton() falhou\n");
    return -1;
  }
  // Enviando mensagem para o servidor
  printf("Enviando mensagem para o servidor: %s [%d]\n", servidor, PORTA);
  sprintf(buf, "Ola, tudo bem?");
  if (sendto(socket_desc, buf, strlen(buf), 0, (struct sockaddr *)&addr_remoto, slen)==-1) {
    perror("Erro ao enviar pacote");
    return -1;
  }
  // Recebendo resposta do servidor
  if ((tam_recebido = recvfrom(socket_desc, buf, TAMBUFFER, 0, (struct sockaddr *)&addr_remoto, &
    slen)) >=0 ){
    buf[tam_recebido] = '\0';
    printf("Mensagem recebida: %s \n", buf);
  }
  close(socket_desc);
  return 0;
}
```

5 Sockets UDP em Java

5.1 Código completo em Java de um servidor UDP

Baixe o código fonte clicando aqui.

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;

/**
 * Servidor UDP imprime a mensagem recebida e depois envia uma resposta ao
 * cliente
 *
 * 2014-08-24
 *
 * @author Emerson Ribeiro de Mello
 */
public class ServidorUDP {

    public static void main(String[] args) throws SocketException, IOException {

        DatagramSocket servidor = new DatagramSocket(1234);
        byte[] dadosRecebidos = new byte[1024];
        byte[] dadosEnviados;

        System.out.println("Ouvindo na porta 1234");
        while (true) {
            DatagramPacket pacoteRecebido = new DatagramPacket(dadosRecebidos, dadosRecebidos.
    length);
            servidor.receive(pacoteRecebido);

            String mensagem = new String(pacoteRecebido.getData());
            System.out.println("Recebido: " + mensagem);
            InetAddress ipOrigem = pacoteRecebido.getAddress();
            int porta = pacoteRecebido.getPort();

            dadosEnviados = "Ola, eu sou o servidor\n".getBytes();
            DatagramPacket pacoteEnviado = new DatagramPacket(dadosEnviados, dadosEnviados.length,
     ipOrigem, porta);
            servidor.send(pacoteEnviado);

        }

    }

}
```

5.2 Código completo em Java de um cliente UDP

Baixe o código fonte clicando aqui.

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;

/**
 * Cliente UDP envia a mensagem ao servidor e imprime a resposta recebida
 * 2014-08-24
 * @author Emerson Ribeiro de Mello
 */
public class ClienteUDP {
    public static void main(String[] args) throws SocketException, UnknownHostException,
    IOException{

        DatagramSocket cliente = new DatagramSocket(4321);
        byte[] dadosRecebidos = new byte[1024];
        byte[] dadosEnviados;
        InetAddress endereco = InetAddress.getByName("localhost");

        String mensagem = "Ola, eu sou o cliente!";

        dadosEnviados = mensagem.getBytes();

        DatagramPacket pacoteEnviado =
                new DatagramPacket(dadosEnviados, dadosEnviados.length,endereco,1234);

        cliente.send(pacoteEnviado);

        DatagramPacket pacoteRecebido = new DatagramPacket(dadosRecebidos, dadosRecebidos.length);
        cliente.receive(pacoteRecebido);

        String recebido = new String (pacoteRecebido.getData());
        System.out.println("Recebido: " + recebido);
        cliente.close();


    }
}
```