



## Laboratório 1: Sockets: Java e C: TCP e UDP

Professor: Emerson Ribeiro de Mello

<http://docente.ifsc.edu.br/mello/std>

### Nota:

*O objetivo deste laboratório: Apresentar como desenvolver aplicações cliente e servidor com Sockets em Java e em C.*

## 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/socket.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.

```
1  /* Criando um socket */
2  // AF_INET = ARPA INTERNET PROTOCOLS -- IPv4
3  // SOCK_STREAM = orientado a conexao
4  // 0 = protocolo padrao para o tipo escolhido -- TCP
5  int socket_desc = socket(AF_INET , SOCK_STREAM , 0);
6
7  if (socket_desc == -1){
8      printf("Nao foi possivel criar socket\n");
9      return -1;
10 }
```

## 2.2 Associando um nome e ouvindo

```
11 //Preparando a struct do socket
12 struct sockaddr_in servidor;
13 servidor.sin_family = AF_INET;
14 servidor.sin_addr.s_addr = INADDR_ANY; // Obtem IP do S.O.
15 servidor.sin_port = htons( 1234 );
16
17 //Associando o socket a porta e endereco
18 if( bind(socket_desc,(struct sockaddr *)&servidor , sizeof(servidor)) < 0){
19     puts("Erro ao fazer bind\n");
20 }
21 puts("Bind efetuado com sucesso\n");
22
23 // Ouvindo por conexoes
24 listen(socket_desc, 3);
```

## Nota:

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

```
25 //Aceitando e tratando conexoes
26 struct sockaddr_in cliente;
27 int c;
28 puts("Aguardando por conexoes...");
29 c = sizeof(struct sockaddr_in);
30 conexao = accept(socket_desc, (struct sockaddr *)&cliente, (socklen_t*)&c);
31 if (conexao<0){
32     perror("Erro ao receber conexao\n");
33     return -1;
34 }
```

## 2.4 Realizando comunicação e fechando o socket

```
35 // pegando IP e porta do cliente
36 char *cliente_ip;
37 int cliente_port;
38 cliente_ip = inet_ntoa(cliente.sin_addr);
39 cliente_port = ntohs(cliente.sin_port);
40 printf("cliente conectou: %s : [%d]\n", cliente_ip, cliente_port);
41
42 // lendo dados enviados pelo cliente
43 if((tamanho = read(conexao, resposta, 1024)) < 0){
44     perror("Erro ao receber dados do cliente: ");
45     return -1;
46 }
47
48 // Coloca terminador de string
49 resposta[tamanho] = '\0';
50 printf("O cliente falou: %s\n", resposta);
51
52 // Enviando resposta para o cliente
53 mensagem = "Ola cliente, tudo bem?";
54 write(conexao, mensagem, strlen(mensagem));
55
56 // fechando
57 close(socket_desc);
58 shutdown(socket_desc, 2);
59 printf("Servidor finalizado...\n");
```

## 2.5 Código completo em C de um servidor 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

   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;
    }
    /******

    //*****comunicacao 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));

    /*****
close(socket_desc);
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 endereço 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.](#)

```
package engtelecom.std.tcp;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
/**
 * Servidor que espera por uma mensagem do cliente (String em UTF) e depois
 * envia uma String de resposta, também em UTF
 *
 * @author Emerson Ribeiro de Mello - 2014-08-24
 */
public class ServidorUTF {

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

        /* Registra serviço na porta 1234 e aguarda por conexões */
        ServerSocket servidor = new ServerSocket(1234);

        System.out.println("Aguardando por conexões");
        Socket conexao = servidor.accept();
        System.out.println("Cliente conectou " + conexao);
        /******
        /* Estabelece fluxos de entrada e saída */
        DataInputStream fluxoEntrada = new DataInputStream(
            new BufferedReader(conexao.getInputStream()));
        DataOutputStream fluxoSaida = new DataOutputStream(conexao.getOutputStream());
        /******
        /* inicia a comunicação */
        String mensagem = fluxoEntrada.readUTF();
        System.out.println("Cliente> " + mensagem);
        fluxoSaida.writeUTF("Oi, eu sou o servidor!");
        /******
        /* Fecha fluxos e socket */
        fluxoEntrada.close();
        fluxoSaida.close();
        conexao.close();
        servidor.close();
    }
}
```

### 3.2 Código completo em Java de um cliente TCP

[Baixe o código fonte clicando aqui.](#)

```

package engtelecom.std.tcp;

import java.io.BufferedInputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;

/**
 * Cliente que envia uma String em UTF por um socket TCP e espera
 * por uma resposta do servidor
 *
 *
 * 2014-08-24
 * @author Emerson Ribeiro de Mello
 */
public class ClienteUTF {

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

        /* Estabelece conexao com o servidor */
        Socket conexao = new Socket(ip,porta);
        System.out.println("Conectado! " + conexao);
        /***/
        /* Estabelece fluxos de entrada e saida */
        DataOutputStream fluxoSaida = new DataOutputStream(conexao.getOutputStream());
        DataInputStream fluxoEntrada = new DataInputStream(
            new BufferedInputStream(conexao.getInputStream()));
        /***/
        /* Inicia comunicacao */
        fluxoSaida.writeUTF("Oi, eu sou o cliente!");
        fluxoSaida.flush();

        String mensagem = fluxoEntrada.readUTF();
        System.out.println("Servidor> " + mensagem);
        /***/
        /* Fecha fluxos e socket */
        fluxoSaida.close();
        fluxoEntrada.close();
        conexao.close();

    }
}

```

### 3.3 Código completo em Java de um cliente TCP que envia e recebe um vetor de bytes

Este cliente é ideal para se comunicar com o servidor escrito em C da listagem presente na Seção 2.5.

Baixe o código fonte clicando [aqui](#).

```
package engtelecom.std.tcp;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.Socket;

/**
 * Cliente que envia um vetor de caracteres por um socket TCP e espera
 * por uma resposta do servidor
 *
 * 2014-08-24
 * @author Emerson Ribeiro de Mello
 */
public class ClienteVetorBytes {

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

        /* Estabelece conexao com o servidor */
        Socket conexao = new Socket(ip,porta);
        System.out.println("Conectado! " + conexao);
        /*****
        /* Estabelece fluxos de entrada e saida */
        OutputStream saida = conexao.getOutputStream();
        BufferedReader entrada = new BufferedReader(new InputStreamReader(conexao.
        getInputStream()));
        /*****
        /* Inicia comunicacao */
        String mensagem = "Oi, eu sou o cliente!";
        saida.write(mensagem.getBytes());
        saida.flush();

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

        /*****
        /* Fecha fluxos e socket */
        saida.close();
        entrada.close();
        conexao.close();
    }
}
```



## 4 Sockets UDP em C

### 4.1 Código completo em C de um servidor UDP

Baixe o código fonte clicando aqui.

```
#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.

```
#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.

```
package engtelecom.std.udp;

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.

```
package engtelecom.std.udp;

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();

    }
}
```

## 6 Exercícios

1. Desenvolva um aplicativo servidor que armazene em memória zero ou mais listas de *String*. O servidor deverá oferecer uma interface de forma a permitir ao cliente criar uma nova lista, adicionar valores em uma lista já existente, obter o último valor adicionado em uma lista, e por consequência, removê-lo dessa lista. Desenvolva um aplicativo cliente capaz de fazer requisições a esse servidor. Cliente e Servidor poderão ser desenvolvidos em diferentes linguagens de programação ou em uma mesma linguagem.
2. Desenvolva um aplicativo cliente C e um aplicativo servidor em Java que permitam a transferência de arquivos (de qualquer tamanho) pela rede. O servidor deverá ser capaz de atender múltiplos clientes. Ao subir o servidor deve-se informar a porta e o caminho do diretório onde estarão os arquivos que serão servidos aos clientes. Ao iniciar o cliente deve-se informar o endereço IP e porta do servidor, bem como o nome do arquivo que deseja transferir. Exemplos:

```
576 java Servidor 1234 /tmp/repositorio
577
578 ./cliente 127.0.0.1 1234 arquivo.txt
```