Comunicação via Sockets

Prof. Orlando Loques - IC/UFF

versão 26-04-2001

Referências:

- Slides Prof. Alexandre Sztajnberg, UERJ
- Capítulo 15, Applied Operating System Concepts, Silberschatz & Galvin; Wiley 2000

Sockets (i)

- *Sockets* definem pontos terminais de acesso à comunicação
- Pares de processos/*threads* comunicam-se através de pares de sockets
- Cada *socket* é definido por um endereço IP concatenado a um número identificador de porta
- Clientes e servidores conectam-se para comunicação através de portas
- Serviços comuns são associados a portas padronizadas (# < 1024), e.g., telnet:23; ftp:21; http:80

Sockets (ii)

- Interações simples
  - rápidas, semântica de melhor esforço: datagrama (UDP)
- Interações longas
  - garantia de entrega, sequenciamento: circuito virtual (TCP)
- A programação de acessos à rede passa a ter uma interface similar a do sistema de arquivos do Unix
- Proposta no BSD UNIX, Univ. Calif. Berkelay
  - interface definida por um socket (tomada / ponto de acesso)

Serviços de Rede (i)

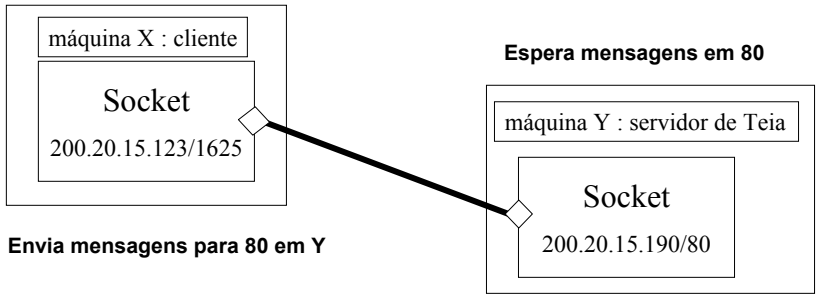
- orientado a conexão/circuito virtual (TCP): um canal lógico é explicitamente estabelecido entre dois processos comunicantes e usado para transmitir sequências de bytes (streams) entre os mesmos; a camada de transporte se encarrega de transferir a sequência de forma correta e confiável
- sem conexão/datagrama (UDP): as mensagens de tamanho fixo (datagramas) são transmitidas individualmente para destinações especificadas; nenhuma garantia de entrega de mensagem é assumida

Serviços de Rede (ii)

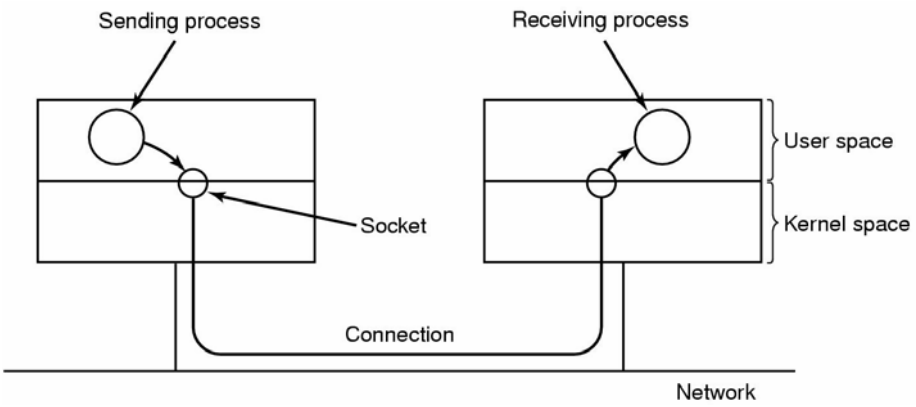
	Service	Example
Connection-oriented ~TCP	Reliable message stream	Sequence of pages of a book
	Reliable byte stream	Remote login
	Unreliable connection	Digitized voice
Connectionless ~UDP	Unreliable datagram	Network test packets
	Acknowledged datagram	Registered mail
	Request-reply	Database query

Sockets (iii)

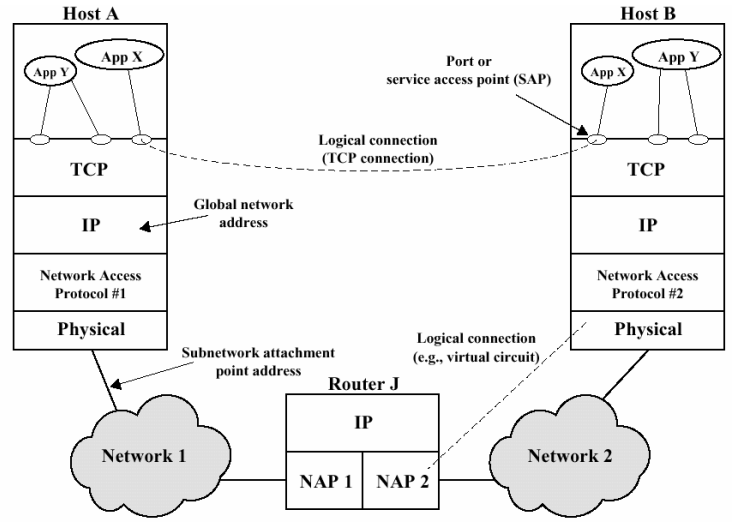
- **Cliente na máquina X (IP: 200.20.15.123) porta 1625 (usuários podem usar qq # > 1024)**
- **Servidor de Teia na máquina Y (IP: 200.20.15.190) e porta 80 (padrão http)**



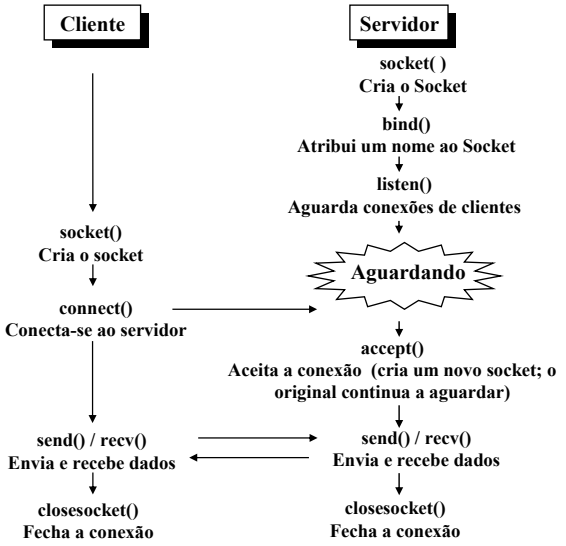
Sockets Unix-BSD (iv)



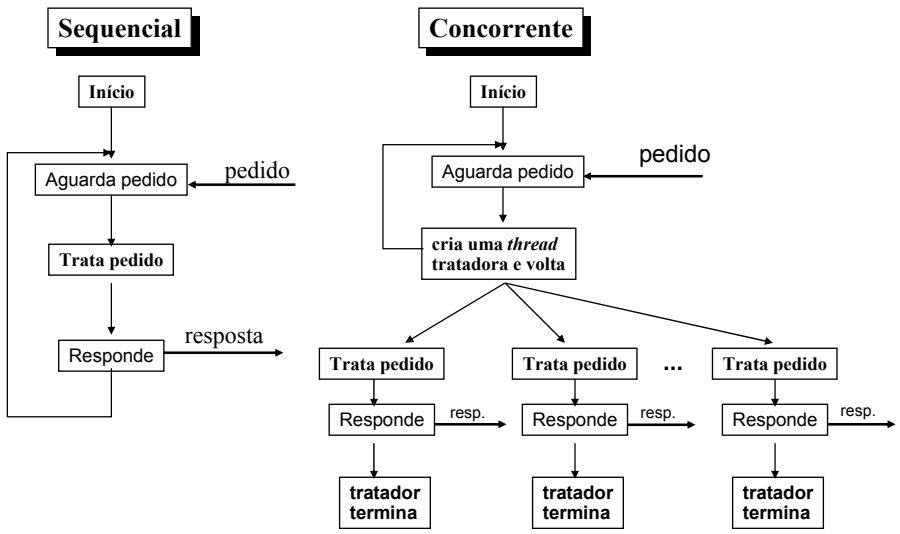
Cliente Servidor usando Middleware



Socket TCP



Servidores Iterativos e Concorrentes



```
/**
 * Java Client Server Socket Communication
 *
 * @author Greg Gagne, Peter Galvin, Avi Silberschatz
 * @version 1.0 - July 15, 1999
 * Copyright 2000 by Greg Gagne, Peter Galvin, Avi Silberschatz
 * Applied Operating Systems Concepts - John Wiley and Sons, Inc.
 */
```

Funcionamento:

O cliente cria uma conexão com o socket do servidor (S) e envia através dela uma "mensagem" para o servidor pedindo a hora/data. O servidor cria um socket para receber as mensagens dos clientes e cria uma thread (chamada Connection) para tratar e responder individualmente cada pedido.

```
/**
 * Client.java - this client accesses the time of day from a server
 * Socket # 127.0.0.1 (lopback) allows client and server to be in the same machine
*/
public class Client
{
        public Client() {
                try {
                        Socket s = new Socket("127.0.0.1",5155);

                        InputStream  in = s.getInputStream();

                        BufferedReader  bin = new BufferedReader(new InputStreamReader(in));

                        System.out.println(bin.readLine());

                        s.close();
                }
                catch (java.io.IOException e) {
                        System.out.println(e);
                        System.exit(1);
                }
        }

        public static void main(String args[]) {
                Client client = new Client();
        }
}
```

```
/**
 * Server.java - This is a time of day server that listens on port 5155.
*/
public class Server
{
        public Server() {
                // create the socket the server will listen to
                try {
                        s = new ServerSocket(5155);
                }
                catch (java.io.IOException e) {
                        System.out.println(e);
                        System.exit(1);
                }

                // OK, now listen for connections
                System.out.println("Server is listening ....");
                try {
                        while (true) {
                                client = s.accept();
                                // create a separate thread  to service the request
                                c = new Connection(client);
                                c.start();
                        }
                }
                catch (java.io.IOException e) {
                        System.out.println(e);
                }
        }
```

```
/**
 * Server.java - This is a time of day server that listens on port 5155
*
* Continuação ...
*/

                public static void main(String args[]) {
                        Server timeOfDayServer = new Server();
                }

                private ServerSocket    s;
                private Socket          client;
                private Connection      c;
        }
```

```
/**
 * Connection.java - This is the separate thread that services each request
*/

public class Connection extends Thread
{
        public Connection(Socket s) {
                outputLine = s;
        }

        public void run() {
                // getOutputStream returns an OutputStream object, allowing ordinary file IO over the socket

                try {
                        // create a new PrintWriter with automatic flushing
                        PrintWriter pout = new PrintWriter(outputLine.getOutputStream(), true);

                        // now send a message to the client
                        pout.println("The Date and Time is " + new java.util.Date().toString());

                        // now close the socket
                        outputLine.close();
                }
                catch (java.io.IOException e) {
                        System.out.println(e);
                }
        }

        private Socket          outputLine;
}
```

Comportamento do objeto Client: client : O usuário solicita o instanciamento do objeto Client:client como um processo, invocando o método main() atraves da linha de comando "java client.class"; Client: utiliza 3 métodos:

Client() = construtor da classe;
main() = método usado pelo SO para instanciar o processo;
println() = método do objeto java.System.out usado para apresentar Strings na saída padrão do sistema hospedeiro.

Passo 0: o método main() invoca o método Client(), que eh o construtor do objeto client;

Passo 1: o método Client() tenta instanciar o objeto Socket:s, que ficará encapsulado pelo objeto Client: client, e tambem tenta estabelecer uma conexão usando o identificador 127.0.0.1:5155;

Passo 2: o método s.getInputStream() tenta ligar o objeto Socket:s ao objeto InputStream:in, que provê os métodos básicos para a entrada de streams no formato do ambiente nativo;

Passo 3: o método BufferedReader() tenta instanciar o objeto BufferedReader:bin, que implementará um buffer de *char[], apos tentar instanciar o objeto da "Anonymous Inner Class" InputStreamReader: <anonymous>, que implementará um conversor *bytes[] >> *char[], que por sua vez receberá como parametro o objeto InputStream:in, que já contem o objeto Socket:s;

Obs: Os "helper objects" InputStreamReader:<anonymous> e BufferedReader:bin tem como função converter streams, de forma que o método BufferedReader.readLine() retorne String.
Ou seja: *byte[] >> *char[] >> string.

Passo 4: Imprime String recebida na saída padrão(console);

Passo 5: Fecha conexão e destroi o objeto Socket:s, que tambem invoca
a destruição de todos os objetos que o encapsulam.

Comportamento do objeto Server:timeOfDayServer: O usuário solicita o instanciamento do objeto Server:timeOfDayServer como um processo, invocando o método main() atraves da linha de comando "java Server.class"; Server:timeOfDayServer : utiliza 2 métodos:

Server() = construtor da classe;
main() = método usado pelo SO para instanciar o processo.

Passo 0 = o método main() invoca o método Server(), que eh o construtor do objeto timeOfDayServer;
Passo 1 = o método Server() tenta instanciar o objeto ServerSocket:s, que ficará encapsulado pelo objeto Server:timeOfDayServer;
Passo 2 = o método s.accept() instancia um daemon nativo que permanece monitorando eventos de tentativa de conexão na porta dada pelo estado identificador 5155;
Passo 3 = na ocorrencia do evento de tentativa de conexão, o objeto ServerSocket:s passa o objeto Socket:client como um member object para o instanciamento do objeto Connection:c que, por ser derivado da classe Thread, será implementado no sistema hospedeiro como um lightweight process.
Passo 4 = Por sua vez o objeto Connection:c repassará o objeto Socket:client como um member object para o instanciamento do objeto PrintWriter:pout e tambem o encapsulará como um member object. O método construtor PrintWriter() usa os parametros client.getOutputStream() e "autoflush" para tentar ligar este objeto ao recurso socket nativo do ambiente hospedeiro, usando a modalidade que automaticamente esvazia o buffer a cada invocação do método println();

Comportamento do objeto Server:timeOfDayServer: continuação

Obs: No caso do objeto Client:client foi diferente. Lá usou-se BufferedReader:bin(InputStreamReader(InputStream:in)). Aqui poderia-se usar o analogo BufferedWriter:x(OutputStreamWriter(OutputStream:y)), porém preferiu-se usar uma classe mais flexivel: PrintWriter. Não existe analogo a esta classe no JDK 1.1 para o ramo Reader da árvore de hierarquia de classes. Esta classe instancia o objeto PrintWriter:pout que automaticamente instancia os objetos BufferedWriter(OutputStreamWriter(OutputStream(Socket:client) para realizar a conversão String >> *char[] >> *byte[].

Passo 5 = o método run() invoca o método pout.println() que foi implicitamente ligado ao método client.getOutputStream(), atraves da ligação realizada no passo 4, de forma a enviar a String resultante de Date().toString() pela rede na forma de um stream de bytes;
Passo 6 = o método run() invoca o método client.close() para fechar a conexão, o que provocará a destruição do objeto Socket:client, que tambem provocará a destruição de todos os objetos que o encapsulam, ou seja: Connection:c(PrintWriter:pout). Observe que o objeto ServerSocket:s permanecerá intacto.

Referências sobre sockets:

http://www.ic.uff.br/javatutor/networking/index.html

http://www.ic.uff.br/javadocs/guide/net/index.html