

Sistemas Operacionais II N

Aula prática

Java Sockets, RPC e RMI

Eduardo Bezerra



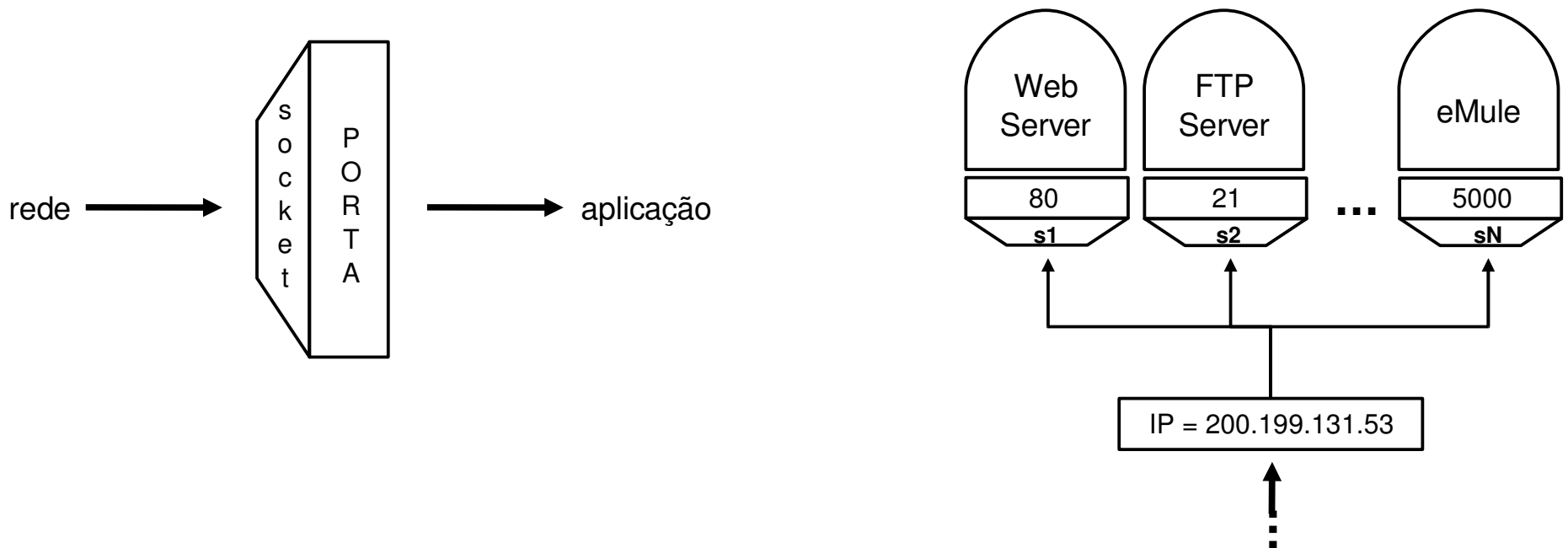
Roteiro da prática

- Revisar alguns mecanismos de comunicação entre processos
 - Sockets (Java)
 - RPC (C)
 - RMI (Java)
- Definir um problema a resolver
 - implementar um servidor de hora e um cliente que o consulte
- Entender a solução com Sockets, RPC e RMI
- Compilar e executar no laboratório

Sockets

■ Definição:

- “Um **socket** é um ponto final de um link de comunicação bidirecional entre dois programas executando em rede. Cada socket está associado a um número de porta de forma que a camada de transporte pode identificar a aplicação a que os dados estão destinados” - java.sun.com





Sockets

UDP



Sockets UDP – User Datagram Protocol

- Maneira simples de se transmitir dados
- Não requer estabelecimento de conexão entre o transmissor e o receptor
- Cada pacote (datagrama) é:
 - Independente de outros pacotes (ordem não é verificada)
 - Sem garantia de chegada
 - Sem garantia de atraso máximo

UDP Sockets em Java – enviando

- Cria-se um socket UDP
 - Pode-se especificar ou não a porta à qual ele estará associado
 - Servirá tanto para enviar quanto para receber mensagens
- Cria-se um pacote UDP (datagrama), atribuindo-lhe:
 - Dados a transmitir
 - Comprimento dos dados
 - Endereço de destino
 - Porta de destino
- Faz um send do pacote através do socket

```
DatagramSocket socket =  
    new DatagramSocket();  
  
DatagramPacket pkt =  
    new DatagramPacket (msg,  
        msg.length, ip_destino,  
        porta);  
  
socket.send(pkt);
```

UDP Sockets em Java – recebendo

- Cria-se um socket UDP
- Cria-se um buffer para ser preenchido com os dados recebidos
- Cria-se um pacote UDP, atribuindo o buffer criado
- Faz um receive no socket passando o pacote vazio
- Pode-se recuperar o endereço e porta de origem para enviar uma resposta

```
DatagramSocket socket =  
    new DatagramSocket(678);
```

```
byte[] buffer = new  
    byte[1024];
```

```
DatagramPacket pkt =  
    new DatagramPacket  
        (buffer, buffer.length);
```

```
socket.receive(pkt);
```

```
pkt.getAddress();  
pkt.getPort();
```

UDP Sockets em Java – exemplo

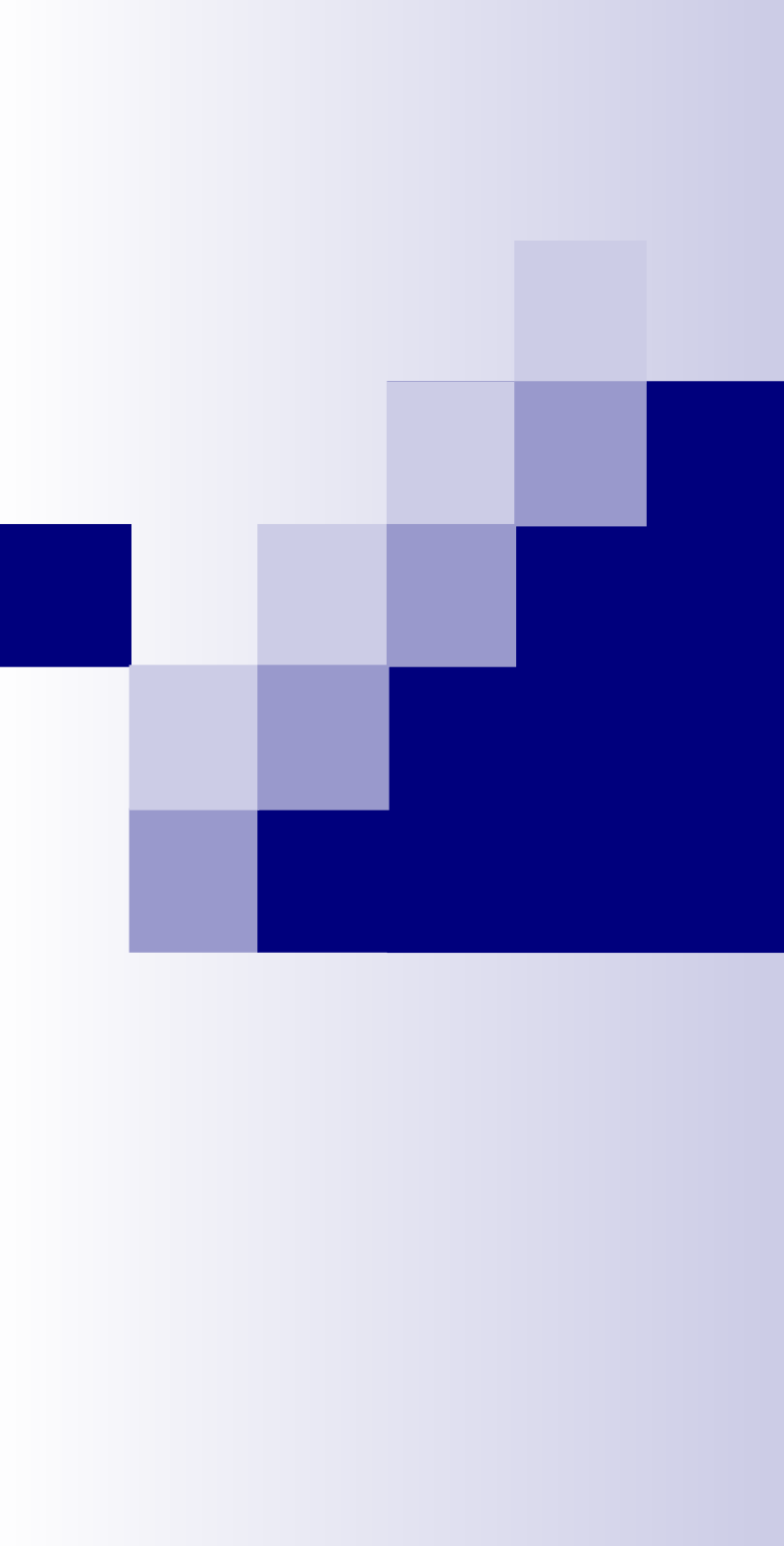
```
$ cd ~
$ wget www.geocities.com/kdubezerra/pratica_sop2_srr.zip
$ unzip pratica_sop2_srr.zip
$ cd pratica_sop2_srr/java_sockets/udp
$ gedit ClienteUDP.java &
$ gedit ServidorUDP.java &
$ javac ClienteUDP.java
$ javac ServidorUDP.java
```

Executando o servidor:

```
$ java ServidorUDP <porta>
```

Executando o cliente:

```
$ java ClienteUDP <endereço> <porta> <mensagem>
$ java ClienteUDP <endereço> <porta> HORA
```

Sockets

TCP



Sockets TCP – Transmission Control Protocol

- Maneira confiável de se transmitir dados
- É estabelecida uma conexão entre os processos
 - Um dos processos espera conexões (servidor), enquanto o outro conecta-se (cliente)
- É garantido o recebimento das mensagens
 - O transmissor espera uma confirmação dentro de um prazo
 - Havendo timeout, a mensagem é retransmitida
- No receptor, as mensagens são entregues na ordem de envio
 - Se as mensagens m1 e m2 são enviadas, mas m2 chega ao destino primeiro, o receptor espera que m1 chegue para entregá-las ao processo na ordem correta
- Abstraído o conceito de pacotes - streaming

TCP Sockets em Java – servidor

- Cria-se um socket do servidor
 - Pode-se especificar a porta
- Espera conexões dos clientes
 - Novo socket para cada cliente
- Associa um objeto de escrita à stream de saída
- Associa um objeto de leitura à stream de entrada
- Envia-se e recebe-se dados como se estivesse lendo e escrevendo em uma stream qualquer:
 - `in.readLine();`
 - `out.println();`

```
ServerSocket ss = new
    ServerSocket(678);

Socket cs = ss.accept();

out = new PrintWriter
    (cs.getOutputStream(), true);

in = new BufferedReader(new
    InputStreamReader
    (cs.getInputStream()));

String recebida = in.readLine();
out.println ("resposta");
```

TCP Sockets em Java – cliente

- Cria-se um socket
 - Passa-se como parâmetros endereço e porta do servidor
 - A criação do socket com estes parâmetros implica na conexão
- Associa um objeto de escrita à stream de saída
- Associa um objeto de leitura à stream de entrada
- Envia-se e recebe-se dados como se estivesse lendo e escrevendo em uma stream qualquer:
 - `in.readLine();`
 - `out.println();`

```
Socket sock = new  
    Socket("time.server.com",  
        678);
```

```
out = new PrintWriter  
    (sock.getOutputStream(),  
    true);
```

```
in = new BufferedReader(new  
    InputStreamReader  
    (sock.getInputStream()));
```

```
String recebida = in.readLine();  
out.println ("resposta");
```

TCP Sockets em Java – exemplo

```
$ cd ~  
$ cd pratica_sop2_srr/java_sockets/tcp  
$ gedit ClienteTCP.java &  
$ gedit ServidorTCP.java &  
$ javac ClienteTCP.java  
$ javac ServidorTCP.java
```

Executando o servidor:

```
$ java ServidorTCP <porta>
```

Executando o cliente:

```
$ java ClienteTCP <endereço> <porta>
```



RPC



RPC – Remote Procedure Call

- RPC é um mecanismo que permite a chamada a procedimentos implementados e executados em máquinas remotas
 - Permite acesso a funcionalidades presentes em computadores remotos
 - Permite paralelização de processamento, utilizando a CPU do computador remoto



RPC – Passos

- Especificar a interface do procedimento remoto
 - Descrita em IDL do RPC
 - Utilizando o comando `rpcgen`, são gerados:
 - Header com a declaração dos procedimentos
 - 2 arquivos `.c` de stubs: lado cliente e lado servidor
- Implementação do procedimento remoto
- Implementação do cliente que faz a chamada ao procedimento remoto

RPC – descrição da interface

- Arquivo .x com descrição em IDL da interface
 - Tipo de retorno
 - Tipos dos parâmetros
- Apenas um parâmetro pode ser passado
- É escrito um bloco program que conterá versões dos procedimentos
 - Cada versão modificar ou adicionar procedimentos
- Cada programa, cada versão e cada procedimento tem um número associado
- Utilizado o comando rpcgen para gerar automaticamente os stubs do cliente e do servidor

```
program MY_PROG{  
    version MY_VERS{  
        long PROC(string) = 1;  
        string TIME(void) = 2;  
    } = 1;  
} = 123456789;
```

RPC – protótipos dos procedimentos

```
//mine.h
//...
extern long * proc_1 (char **, CLIENT *);
extern long * proc_1_svc (char **, struct svc_req *);
//...
```

- Após o comando `rpcgen`, pegar do header gerado o protótipo dos procedimentos com `_svc` e definí-los no código do servidor
- Utilizar o protótipo sem `_svc` para fazer as chamadas no código do cliente

RPC – implementação

- Retorno e parâmetros são passados como ponteiros
- Variável a retornar tem que ser `static`

```
long * proc_1_svc(char ** str,  
    struct svc_req * req) {  
  
    static long tamanho;  
  
    tamanho = strlen(*str);  
  
    return (&tamanho);  
  
}
```

RPC – invocação

- É criado um cliente, informando:

- ☐ Endereço do servidor
- ☐ Código do programa
- ☐ Código da versão
- ☐ Protocolo utilizado

- A chamada é feita passando o parâmetro e o cliente como argumentos

- O retorno é um ponteiro

```
int main() {  
    CLIENT *cl =  
        clnt_create(end_servidor,  
        MY_PROG, MY_VERS, "udp");  
  
    char msg[10] = "mensagem";  
  
    long * tam = proc_1(&msg, cl);  
  
    printf (" %ld \n ", *tam);  
  
    return 0;  
}
```

RPC – exemplo

```
$ cd ~
$ cd pratica_sop2_srr/rpc
$ ls
$ gedit date.x &
$ rpcgen date.x
$ ls
$ gedit date.h &
$ gedit date_server.c &
$ gedit date_client.c &
$ gcc -o server date_server.c date_svc.c -lrpcsvc
$ gcc -o client date_client.c date_clnt.c -lrpcsvc
```

Executando o servidor:

```
$ ./server
```

Executando o cliente:

```
$ ./client <endereço>
```



RMI



RMI – Remote Method Invocation

- Semelhante ao RPC
- Orientado a objetos
- RMI é um mecanismo que permite a invocação de métodos de objetos remotos
 - Permite acesso a funcionalidades presentes em computadores remotos
 - Permite paralelização de processamento, utilizando a CPU do computador remoto



RMI – Passos

- Especificar a Interface do objeto remoto
- Implementação do objeto remoto
 - Classe que implementará a Interface
- Implementação do objeto servidor
 - Nele haverá o método `main()`
 - Ele mesmo pode ser o obj. remoto se implementar a Interface
- Implementação do cliente que faz a chamada ao procedimento remoto

RMI – Interface Remota

- Deve estender a classe `java.rmi.Remote`

- Cada método deve ter em seu cabeçalho `throws java.rmi.RemoteException`

```
public interface
    MyRemoteInterface extends
        java.rmi.Remote {

        public String metodo()
            throws
                java.rmi.RemoteException;

    }
```

RMI – Classe com implementação

- Deve implementar a Interface definida no passo anterior
- Deve estender `java.rmi.server.UnicastRemoteObject`
- O construtor chama o construtor do pai

```
public MyImpl implements
    MyRemoteInterface extends
    java.rmi.server.
    UnicastRemoteObject {

    public MyImpl() {
        super();
    }

    public String metodo() {
        return "texto";
    }

}
```

RMI – Servidor

- Deve instanciar o objeto cujo método será invocado
- O objeto deve ser disponibilizado para acesso remoto com o método `rebind()` do RMI
- Deve estar dentro de um bloco `try{ }catch{ }` ou utilizar a cláusula `throws`

```
try {  
    MyImpl obj = new MyImpl();  
  
    java.rmi.Naming.rebind("rem_obj",  
obj);  
} catch (Exception e) { //tratar  
}
```

Para poder receber como parâmetros objetos de classes do cliente (o servidor faz download dos .class), é necessário usar o `RMISecurityManager`.

RMI – Cliente

- Pega uma referência do tipo da Interface e a faz apontar para o objeto remoto
- Os métodos são invocados como se fossem locais
- Deve estar dentro de um bloco try{ }catch{ } ou utilizar a cláusula throws

```
try {  
  
    MyRemoteInterface remoto =  
        (MyRemoteInterface)  
        Naming.lookup("rmi://host/rem_obj");  
  
    String retorno = remoto.metodo();  
  
} catch (Exception e) { //tratar  
}
```

RMI – Exemplo

```
$ cd ~/pratica_sop2_srr/rmi/server && ls
$ gedit HoraInterface.java &
$ gedit HoraImpl.java &
$ gedit HoraServer.java &
$ javac HoraServer.java
$ cd ~/pratica_sop2_srr/rmi/client && ls
$ gedit HoraClient.java &
$ javac HoraClient.java
```

Executando o servidor (no diretório .../server):

```
$ java HoraServer
```

Executando o cliente (no diretório .../client):

```
$ java HoraClient rmi://<endereço>/hora
```

Perguntas?

```
#include <stdio.h>
int main(void)
{
    int count;

    for (count = 1; count <= 500; count++)
        printf("I will not throw paper airplanes in class.");
    return 0;
}
```

AMCHD 10-3

