FATEC

CENTRO PAULA SOUZA
COMPETÊNCIA EM EDUCAÇÃO PÚBLICA PROFISSIONAL
Faculdade de Tecnologia Prof. Jessen Vidal
São José dos Campos

GOVERNO DO ESTADO DE SÃO PAULO

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Disciplina: Sistemas Operacionais I

Aula 09: Processos P4

Prof. Diogo Branquinho Ramos
diogo.branquinho@fatec.sp.gov.br
São José dos Campos - SP

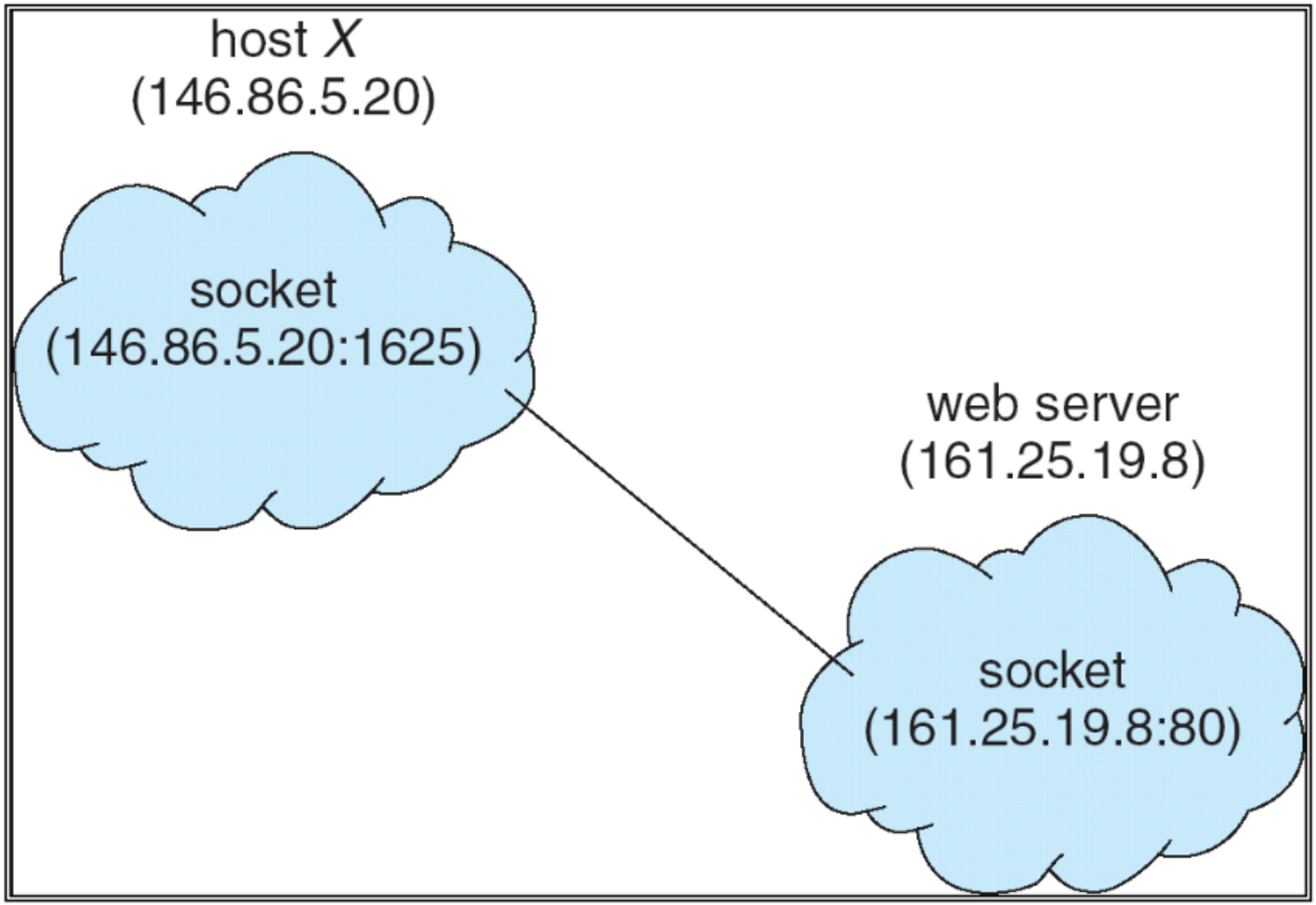
Roteiro

- Modelos em Sistemas cliente-servidor
  - Socket
  - RPC
  - RMI

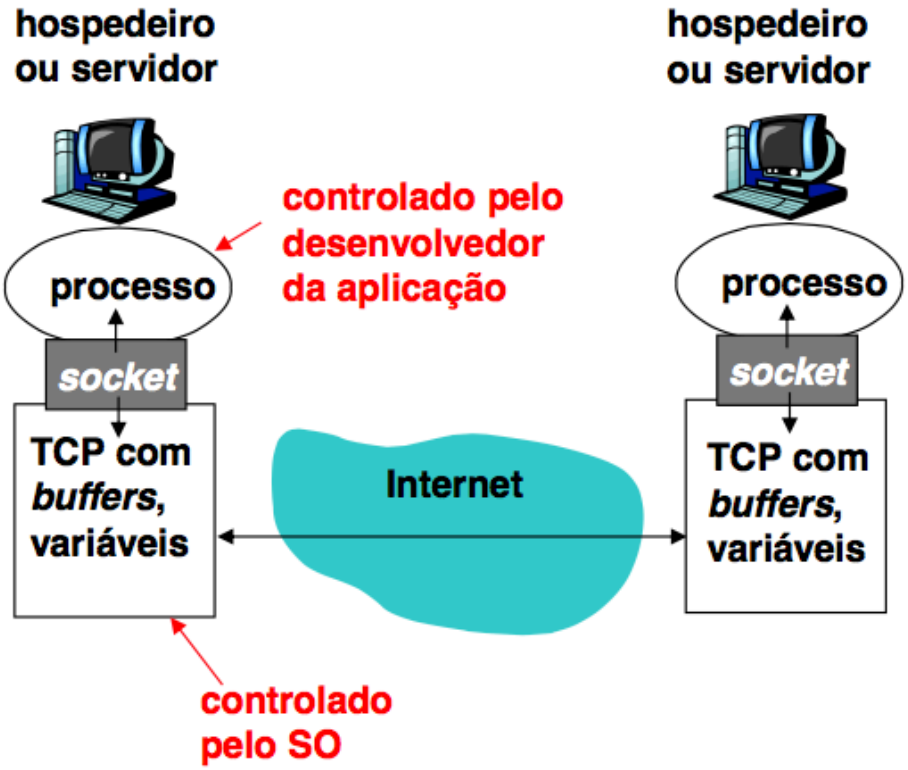
Sockets

- **Definição**
  - Ponto de ligação (oferecido pela API) entre a aplicação e a rede em cada *host*.
  - Permite a identificação exclusiva da aplicação na rede.
    - Mapeia uma porta com um IP.
    - Socket **161.25.19.8:80** – porta **80** no host **161.25.19.8.**
- **Como ocorre a troca de dados**
  - A comunicação acontece entre um par de *sockets*.
  - A troca de dados é sem estrutura.
    - Permite que um fluxo de bytes não-estruturado seja trocado entre os processos em comunicação: é responsabilidade da aplicação impor uma estrutura sobre os dados.

Sockets


host X
(146.86.5.20)
socket
(146.86.5.20:1625)
web server
(161.25.19.8)
socket
(161.25.19.8:80)

Comunicação por socket



Portas bem conhecidas: serviços-padrão

- Telnet: 23
- FTP: 20 e 21
- Web: 80 ou 443
- DNS: 53
- SSH: 22
- NTP: 123
- SMTP: 25 ou 465 ou 587
- IMAP: 143 ou 993
- POP3: 110
- DHCP: 67 e 68
- Netbios: 137 ou 138 ou 139
- SNMP: 161

Testar o comando: netstat -an

Veja mais detalhes em: /etc/services

Pesquisar um pouco sobre esses serviços!

Sockets

- **Funcionamento**
  - Processo cliente inicia a requisição para uma conexão
    - Atribuída a uma porta maior que 1024.
  - O servidor aceita a conexão do socket do cliente.
  - Inicia a troca de dados.
- **Características**
  - Os servidores implementam serviços específicos (telnet, ftp, http...).
    - Em geral, nas portas “bem conhecidas” (1-1024).
  - Toda conexão é exclusiva
    - Garante que todas as conexões consistem em um par de sockets exclusivo.

Comunicação por socket

- Cliente - UDP

```
import socket
HOST = '192.168.1.10'  # Endereco IP do Servidor
PORT = 5000            # Porta do Servidor
udp = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
dest = (HOST, PORT)
print 'Para sair use CTRL+X\n'
msg = raw_input()
while msg <> '\x18':
    udp.sendto (msg, dest)
    msg = raw_input()
udp.close()
```

Comunicação por socket

- Servidor - UDP

```
import socket
HOST = ''                 # Endereco IP do Servidor
PORT = 5000               #Porta do Servidor
udp = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
orig = (HOST, PORT)
udp.bind(orig)
while True:
    msg, cliente = udp.recvfrom(1024)
    print cliente, msg
udp.close()
```

Comunicação por socket

- Cliente - TCP

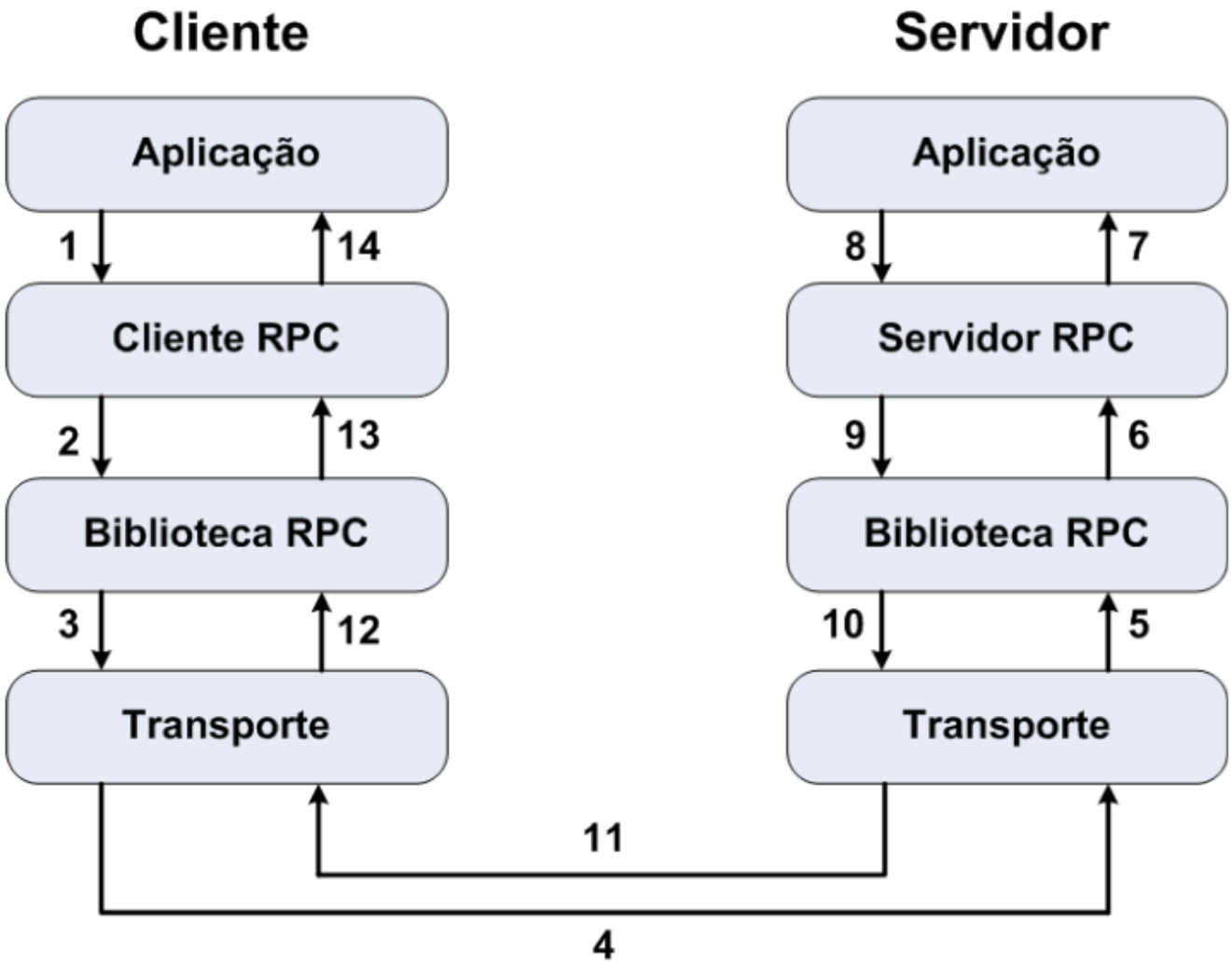
```
import socket
HOST = '127.0.0.1'      # Endereco IP do Servidor
PORT = 5000             # Porta do Servidor
tcp = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
dest = (HOST, PORT)
tcp.connect(dest)
print 'Para sair use CTRL+X\n'
msg = raw_input()
while msg <> '\x18':
    tcp.send (msg)
    msg = raw_input()
tcp.close()
```

Comunicação por socket

- Servidor - TCP

```
import socket
HOST = ''                 # Endereco IP do Servidor
PORT = 5000               # Porta do Servidor
tcp = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
orig = (HOST, PORT)
tcp.bind(orig)
tcp.listen(1)
while True:
    con, cliente = tcp.accept()
    print 'Concetado por', cliente
    while True:
        msg = con.recv(1024)
        if not msg: break
        print cliente, msg
    print 'Finalizando conexao do cliente', cliente
    con.close()
```

Chamadas de Procedimento Remoto



Chamadas de Procedimento Remoto

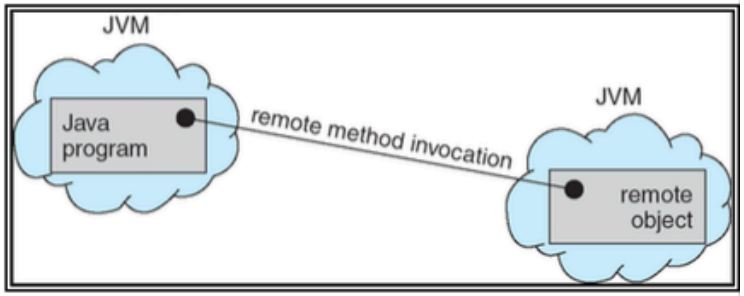
- **Comunicação baseada em mensagens**
  - Passa chamadas de procedimento entre processos nos sistemas em rede.
- **São mais alto nível em relação aos sockets**
  - As mensagens RPCs são estruturadas, não apenas bytes.
- **Funcionamento**
  - A mensagem é endereçada a um serviço RPC escutando em uma porta no sistema remoto e contém um identificador da função a ser executada.
  - A função é executada conforme requisitada e a saída é enviada de volta ao requisitante em uma mensagem separada.
  - Utiliza os sockets.

Chamadas de Procedimento Remoto

- **A implementação do método não está visível**
  - Permite que um cliente chame um procedimento remoto da mesma forma como chamaria um procedimento local.
- **Podem falhar devido a erros comuns de rede**
  - Podem ser duplicadas e executadas mais de uma vez.
- **Outras implementações**
  - CORBA: RPC independente de plataforma.
  - SunRPC: RPC para UNIX e Linux.
  - DCOM: RPC para Windows (atual: .Net Remoting).
  - SOAP: RPC para *Web Service*.
  - RMI: RPC para Java.

Invocação de Método Remoto (RMI)

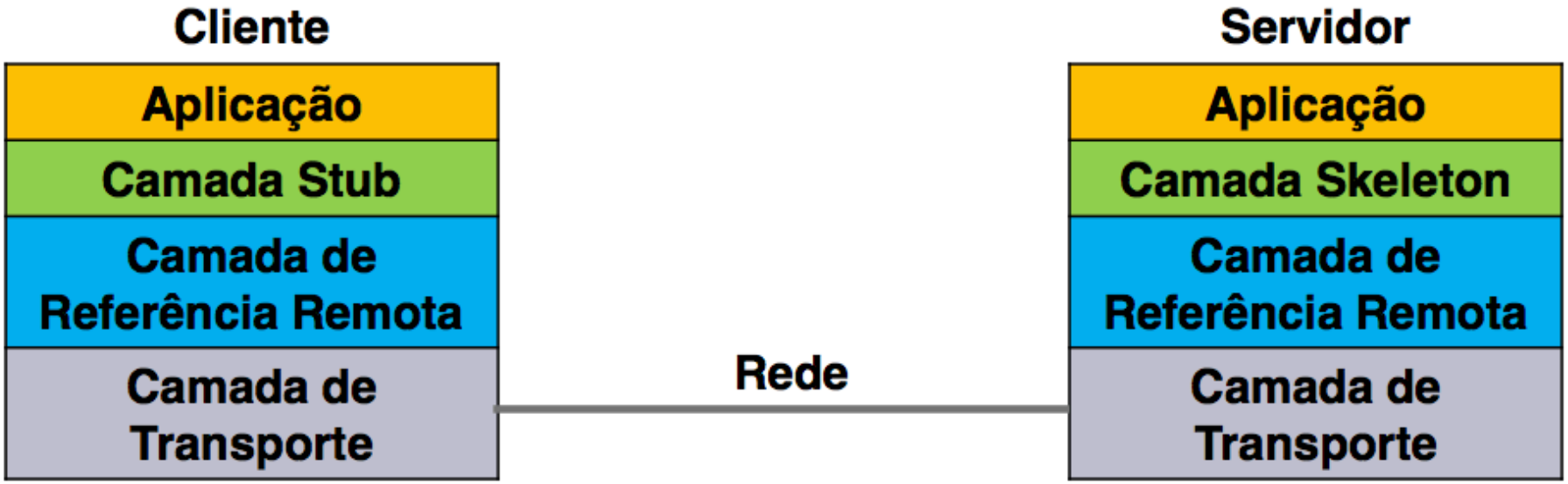
- **RMI permite que um programa Java em uma máquina chame um método em um objeto remoto.**
  - Objetos em JVMs diferentes.
  - Uma das abordagens do JAVA para aplicação distribuída.
- **Abstração RMI**
  - Programadores invocam métodos remotos da mesma maneira que invocam métodos locais.
  - Permite a passagem de objetos.



Esquema: camadas

- **Camadas Stub e Skeleton (“proxies”)**
  - Permite que o cliente e o servidor se comportem como se os objetos, com quais eles estejam lidando, fossem locais.
- **Camada de Referência Remota**
  - Interpreta e gerencia referências feitas de clientes a objetos de serviço remoto.
- **Camada de Transporte**
  - Pode usar diferentes tipos de protocolos: TCP ou UDP.

Esquema: camadas



Exemplo de RMI

```
public interface RemoteDate extends Remote
{
   public abstract Date getDate() throws RemoteException;
}
```

Exemplo de RMI

Essa extensão permite a criação de um único objeto remoto que escuta as requisições da rede

```
public class RemoteDateImpl extends UnicastRemoteObject
      implements RemoteDate
{
   public RemoteDateImpl() throws RemoteException { }

   public Date getDate() throws RemoteException {
      return new Date();
   }

   public static void main(String[] args) {
      try {
         RemoteDate dateServer = new RemoteDateImpl();

         // Bind this object instance to the name "DateServer"
         Naming.rebind("DateServer", dateServer);
      }
      catch (Exception e) {
         System.err.println(e);
      }
   }
}
```

Implementa a interface

Retorno da data

Instância do objeto associada ao nome "DateServer"

Exemplo de RMI

```
public class RMIClient
{
   public static void main(String args[]) {
      try {
         String host = "rmi://127.0.0.1/DateServer";

         RemoteDate dateServer = (RemoteDate)Naming.lookup(host);
         System.out.println(dateServer.getDate());
      }
      catch (Exception e) {
         System.err.println(e);
      }
    }
}
```

Recebendo uma referência do proxy ao objeto registrado no servidor

Chamada do método