Sistemas Distribuídos

Comunicação e sincronização entre processos

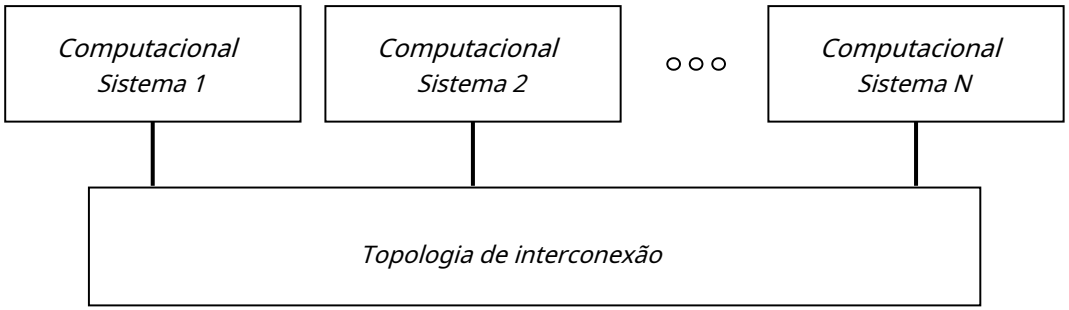
Passagem de mensagens

António Rui Borges

Resumo

- *Sistema de comunicação*
- *Interface de programação*
  - *Protocolo TCP*
  - *Protocolo UDP*
- *Transformação de uma solução concorrente em uma solução distribuída de passagem de mensagens*
  - *Princípios de transformação*
  - *Estrutura da mensagem*
  - *Arquitetura do cliente*
  - *Arquitetura do servidor*
- *Leitura sugerida*

Sistema de comunicação - 1



Os parâmetros que afetam o desempenho de um sistema de comunicação são

- *latência*–atraso que ocorre após a execução de um *enviar* operação e início da recepção de dados (pode ser considerada como a transferência de uma mensagem vazia)
- *taxa de transferência de dados*–velocidade de transmissão de dados entre o remetente e o destinatário
- *largura de banda*–sistema *Taxa de transferência* (volume de tráfego de mensagens por unidade de tempo).

tempo de transmissão da mensagem = latência + comprimento / taxa de transferência de dados

Sistema de comunicação - 2

Quando considerando *multimídia em tempo real* aplicações, há outra propriedade relevante que desempenha um papel importante, *o que* é a *qualidade de serviço*. Ele descreve o sistema capacidade de atender *prazo final* restrições impostas pela transmissão e processamento de fluxos de dados em contínuo. Para que estas operações ocorram de forma satisfatória, é necessário um limite superior de latência e um limite inferior de largura de banda dos canais de dados associados.

Os sistemas de comunicação atuais são bastante confiáveis. As falhas geralmente estão relacionadas a erros no software do remetente ou do receptor e não a erros de rede. Assim, é prática comum transferir para as aplicações a responsabilidade de tratar da detecção e correção dos erros remanescentes, procedimento que é conhecido como *argumento de ponta a ponta*.

Sistema de comunicação - 3

Do ponto de vista do programador de aplicações, o sistema de comunicação deve ser visto de uma forma integrada e abstrata, mascarando a complexidade subjacente das diversas redes físicas que abrange.

Assim, o software de rede é organizado em uma hierarquia de camadas. Cada camada apresenta uma interface operacional para as camadas acima dela que descreve as propriedades do sistema de comunicação neste nível de uma forma lógica. Uma camada é representada por um módulo de software presente em todo sistema computacional conectado à rede.

Cada módulo parece comunicar-se diretamente com o módulo correspondente em outro sistema de computador, mas na realidade os dados não são transmitidos diretamente entre os módulos em cada nível. Cada camada do software de rede se comunica por meio de chamadas de procedimentos locais com os níveis acima e abaixo dela.

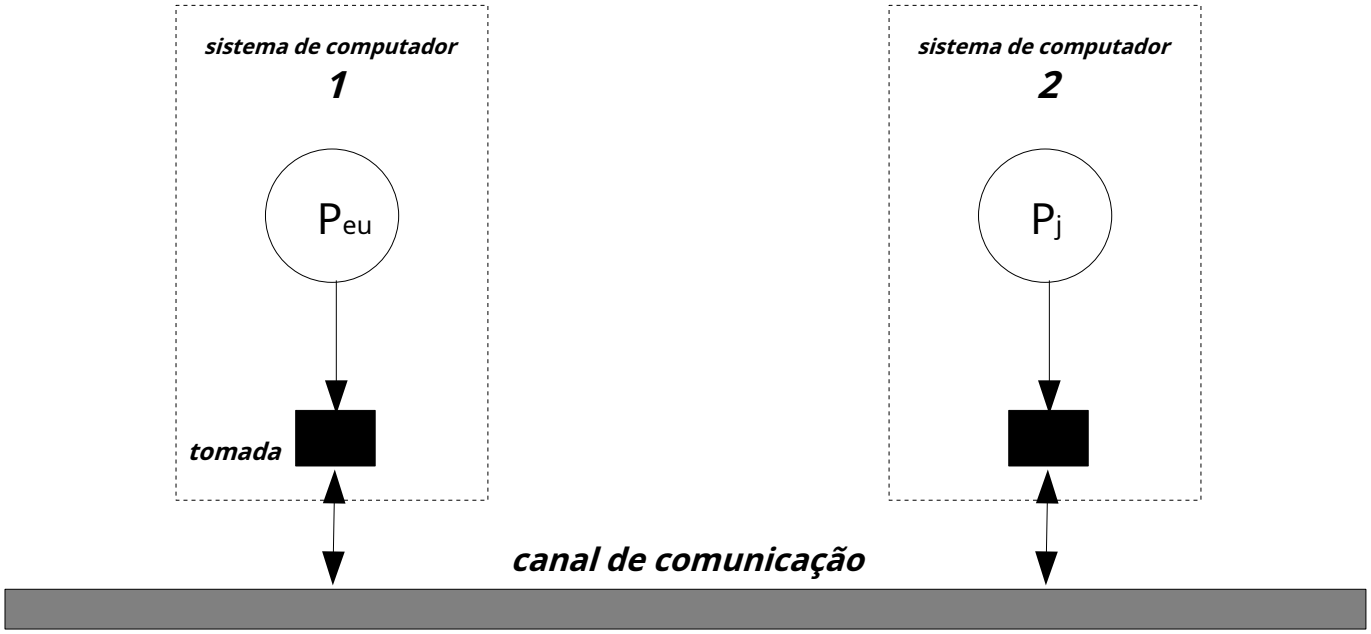
No*lado de envio*, cada camada, exceto a superior, aceita dados em um formato bem definido da camada acima e aplica um procedimento de transformação para encapsulá-los em outro formato bem definido antes de passá-los para a camada abaixo.*Do lado receptor*, as transformações inversas são aplicadas aos dados à medida que atravessam as camadas na direção inversa.

Sistema de comunicação - 4

Modelo OSI

Aplicativo	protocolos projetados para atender aos requisitos de comunicação de aplicações específicas
Apresentação	protocolos para permitir a transmissão de dados em uma representação específica de rede que é independente de hardware
Sessão	protocolos para detecção de erros e recuperação automática
Transporte	protocolos para permitir o endereçamento de mensagens para portas de comunicação associadas a processos
Rede	protocolos para permitir a transferência de pacotes de dados entre os nós da rede
Link de dados (lógico)	protocolos para permitir a transmissão de dados entre nós de rede conectados pelo mesmo link físico
Físico	especificação dos circuitos e sinais que acionam um link físico específico

Interface de programação – 1



O middleware apresenta ao programador da aplicação um dispositivo chamado *ponto final da comunicação* ou *tomada*, para permitir a troca de mensagens entre processos que não compartilham um espaço de endereçamento.

tomadas são caracterizados pela *endereço de IP* do sistema informático e um *porta* que define dentro do sistema de computador o ponto final de um canal de comunicação específico.

Interface de programação – 2

Existem dois protocolos principais para troca de mensagens

- *TCP*–é um*protocolo orientado a conexão*, o que significa que um canal de comunicação virtual deve ser estabelecido entre os pontos finais antes que qualquer troca de dados possa ocorrer
  permite*comunicação bidirecional*porque, uma vez estabelecido o canal, um fluxo de dados pode fluir de cada ponto final
  isso é*assimétrico*, por ter sido projetado especificamente para o modelo cliente-servidor, assume um papel diferente para cada terminal
- *UDP*–é um*sem conexão*protocolo, o que significa que nenhuma comunicação virtual é necessária para que a troca de dados ocorra
  só permite a comunicação unidirecional porque pressupõe a transmissão de uma única mensagem de um dos pontos finais para o outro.*simétrico*porque nenhuma função diferente é atribuída a nenhum dos pontos finais.

Protocolo TCP – 1

O protocolo TCP requer dois tipos de soquetes

- *tomada de escuta*–instanciado pelo*servidor*e onde está escutando uma solicitação de conexão de um*cliente*
- *tomada de comunicação*–instanciado pelo*cliente*quando necessita de uma troca de dados com o*servidor*, e pelo*servidor*quando estabelece um canal de comunicação virtual com o*cliente*.

Protocolo TCP – 2

Lado do cliente

instanciarComSocket(); connectToServer (servidorPublicAdd); openInputStream();

openOutputStream();
escreverRequest();
lerResposta();
closeOutpoutStream();
closeInputStream();
closeComSocket();

Lado do servidor

Base de rosca

instanciarListenSocket (serverPublicAdd);

enquanto (verdadeiro)
{ comSocket =
listenToClientConnectionReq (servidorPublicAdd);
instanciarServiceProxyAgent (comSocket) startServiceProxyAgent ();
}

Agente proxy de serviço

openInputStream();
openOutputStream();
readRequest();
execução local();
escreverResposta();
closeOutpoutStream();
closeInputStream();
closeComSocket();

Protocolo UDP – 1

O protocolo UDP requer um único tipo de soquete para transmitir uma mensagem, chamado *pacote de datagrama*, da origem ao ponto de destino

- *tomada receptora*–instanciado pelo*receptor*em uma porta específica para pacotes recepção de diferentes fontes
- *soquete de envio*–instanciado pelo*remetente*para transmissão de pacotes para diferentes destinos.

Protocolo UDP – 2

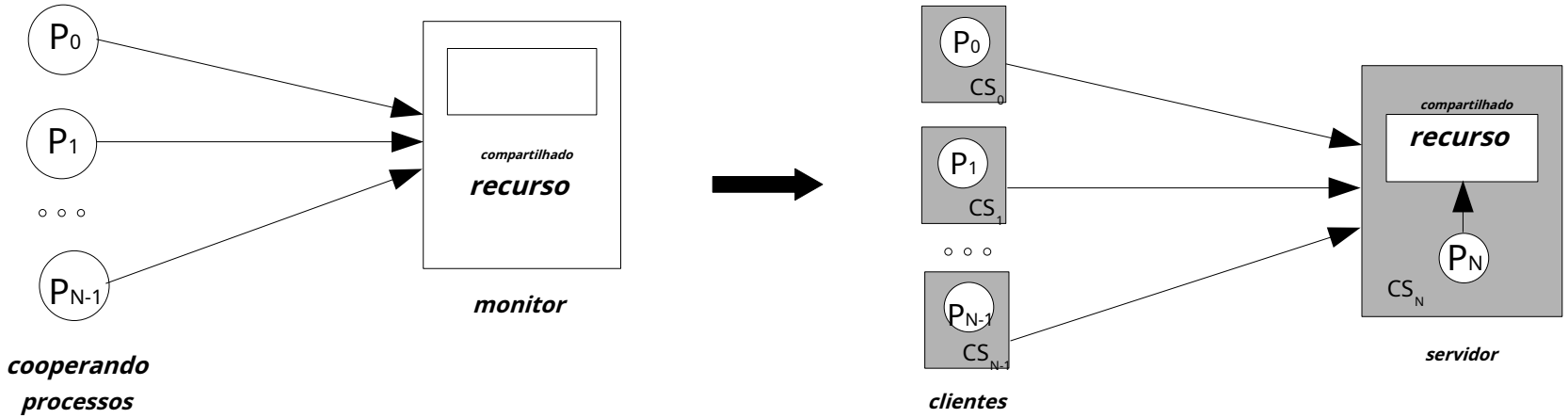
Lado da fonte

instanciarSendSocket();
convMesgToByteArray
(mensagem, byteArray);
instanciarDataPacket
(byteArray, DestPublicAdd); enviar
(pacote de dados);

Lado do destino

instanciarRecSocket
(DestPublicAdd);
receber (dataPacket);
convByteArrayToMesg
(byteArray, mensagem);

Princípios de transformação – 1



Princípios de transformação – 2

Uma característica fundamental da transformação é que esta deve ser realizada com alterações mínimas do código concorrente, ou seja, todo o mecanismo de interação entre as diferentes entidades deve ser mantido tal como foi previamente definido.

Dado que tanto os processos cooperantes como os recursos partilhados residem em sistemas informáticos diferentes, não existe partilha de espaço de endereçamento, o que implica que

- a invocação do método no recurso compartilhado deve ser realizada através da troca de mensagens: uma mensagem para a chamada e outra para o retorno da chamada
- além dos parâmetros do método e do valor de retorno, as mensagens devem incluir os atributos do processo chamador que são relevantes para a execução do método, ou que são alterados pela sua execução
- todos os parâmetros da mensagem devem ser passados por valor.

Estrutura da mensagem – 1

Uma mensagem é transmitida através de um canal de comunicação e, no nível mais baixo, pode ser vista como uma matriz de bytes. Como os processos cliente e servidor são programas separados, é crucial que o receptor saiba como interpretar esse array de bytes e como construir a partir desses dados os valores dos parâmetros da mensagem.

Assim, o conteúdo de uma mensagem deve incluir não apenas os valores dos parâmetros, mas também o seu tipo e como estão estruturados. A operação de construção de uma mensagem com essas características é chamada *empacotamento de informações*, e a operação oposta de recuperar os valores dos parâmetros da matriz de bytes como *desempacotamento de informações*.

Em Java, o empacotamento e o desempacotamento de informações ficam ocultos ao programador. É necessário apenas que o tipo de dados da mensagem seja definido como implementando o `Serializável` interface.

Estrutura da mensagem – 2

```
importar java.io.Serializable;

aula pública Mensagem implementos Serializável {

        final estático privado longo serialVersionUID = <literal longo>;

        /* definição dos parâmetros da mensagem */

        /* instanciação da mensagem */

        /* métodos públicos para obter os valores dos parâmetros da mensagem */
}
```

- se um parâmetro de mensagem for de um tipo de dados de referência, ele também deverá implementar o Serializável interface
- esta regra deve ser aplicada de forma recursiva para que o que reste sejam parâmetros de tipos de dados primitivos

Estrutura da mensagem – 3

```
importarjava.io.Serializable;

aula públicaRegistroimplementosSerializável {

        final estático privado longoserialVersionUID = 20140404L;

        público internonEmp;
        String públicanome;

        públicoRegistro (internonEmp,Cordanome) {
              esse.nEmp = nEmp;
              esse.nome = nome;
        }
}
```

representação serial de "Ana Francisca")

ACED0005737200065265636F726400000000133517402000249000 46E456D704C00046E6F6D6574
00124C6A6176612F6C616E672F537472696E673B7870 0000006974000D416E61204672616E636973 6361

correrMostrarPrincipalNa embalagemmostrarSerializaçãopara ver a interpretação da matriz de bytes acima

Arquitetura do cliente – 1

Na solução concorrente, o principal thread instanciou os processos cooperantes e o recurso compartilhado. Agora, não é mais possível instanciar o recurso compartilhado, pois

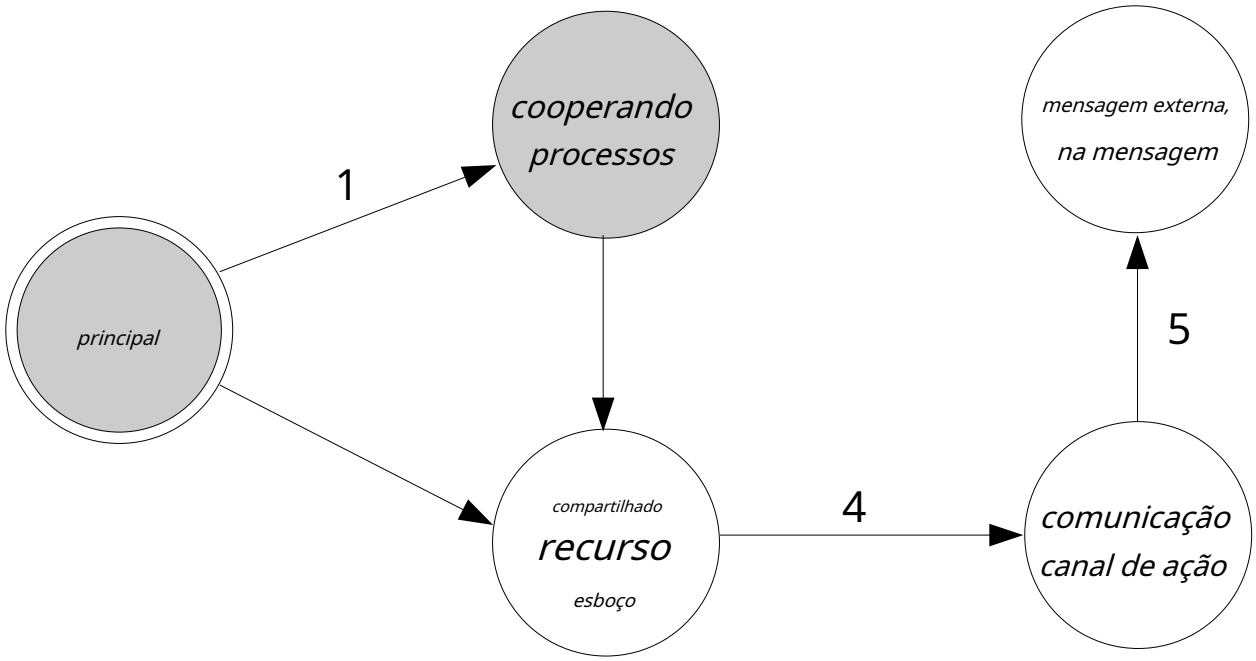
pertence

ente adicionar espaço refeito. No entanto, para manter a maior parte do código inalterada, uma referência remota ao recurso compartilhado será instanciada. Esta referência remota, geralmente chamada de *esboço* do recurso compartilhado, tem como parâmetros de instanciação o endereço de internet do servidor onde está localizado o recurso compartilhado e o número da porta de escuta. Todos os outros valores usados anteriormente para sua instanciação devem agora ser passados através da invocação de um novo método no stub.

para uma diferença

É responsabilidade do stub converter todas as invocações de método no recurso compartilhado em uma troca de mensagens com o servidor onde o recurso compartilhado está localizado.

Arquitetura do cliente – 2



1 – instanciar, iniciar, ingressar

2 – instanciar, possível para-
comunicação do medidor para
inicialização, desligamento

3 – métodos já definidos

4 – instanciar, abrir, fechar, escreverObject, lerObject

5 – instanciar, obter valores de campo

Arquitetura do cliente – 3

Tipo de dados que define o thread principal

Permanece praticamente inalterado. As modificações são as seguintes

- o stub do recurso compartilhado é instanciado em vez do próprio recurso compartilhado
- se outros valores eram anteriormente necessários para a instanciação do recurso compartilhado, esses valores agora são passados através da invocação de um novo método no stub
- se o desligamento do servidor for necessário após o término das operações, será necessária a invocação de um novo método no stub.

Tipo de dados que define os processos cooperantes

Permanece praticamente inalterado. As modificações são as seguintes

- uma referência ao stub do recurso compartilhado é passada na instanciação, em vez de uma referência ao próprio recurso compartilhado.

Arquitetura do cliente – 4

Tipo de dados que define o stub do recurso compartilhado

É novo e deve ser criado. É bastante regular e, para as operações que nele são invocadas, devem ser definidos os seguintes passos

- um canal de comunicação com o servidor é aberto (instanciado)
- uma mensagem de saída é instanciada com base na identificação do método, seus parâmetros e os valores dos atributos do processo chamador que são relevantes para a execução do método
- a mensagem de saída (solicitação de serviço) é enviada
- uma mensagem recebida (resposta) é recebida e verificada quanto à exatidão
- aqueles atributos do processo chamador que foram afetados pela execução do método devem ser atualizados
- o canal de comunicação está fechado
- o método retorna.

Arquitetura do cliente – 5

Tipo de dados que define o canal de comunicação

É novo e deve ser criado. Sua principal característica é encapsular as operações realizadas nos soquetes. O tipo de dados, que é fornecido no exemplos, pode ser usado tal como está ou ser modificado de acordo com requisitos específicos.

Tipo de dados que define a mensagem

É novo e deve ser criado. Pode haver um único tipo de dados que englobe todos os casos ou vários tipos de dados adequados para diferentes situações.

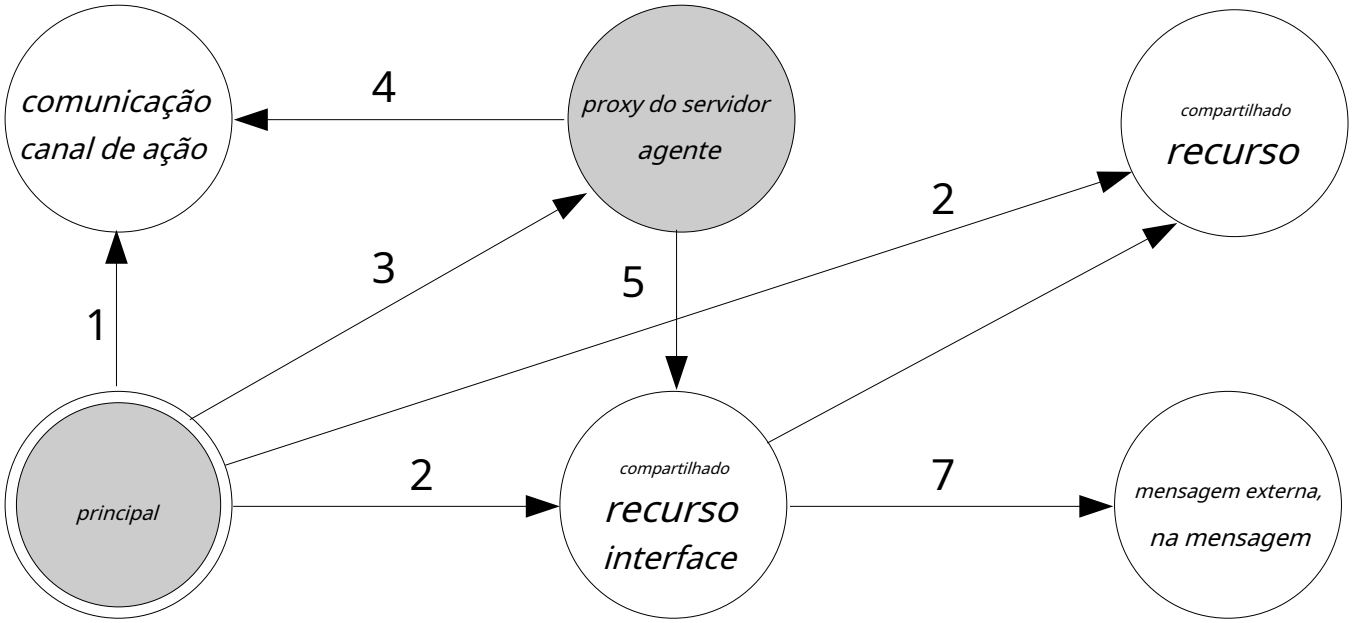
Arquitetura do servidor – 1

O recurso compartilhado é uma entidade passiva. Assim, para que ele esteja operacional, o thread principal, ou thread base, do servidor deve instanciá-lo, bem como um canal de comunicação escutando no endereço público as solicitações de serviço.

Quando chega uma solicitação de serviço, o thread base precisa instanciar e iniciar um *agente proxy de serviço*thread para lidar com a solicitação e retorna próximo à atividade de escuta para cuidar de possíveis novas solicitações de serviço recebidas (*variante de replicação de servidor*).

O agente proxy de serviço recebe a mensagem recebida, decodifica-a e configura-se como um clone do cliente incorporando os atributos do processo necessários, invoca o método correspondente no recurso compartilhado, compõe a mensagem de saída, envia-a, fecha o canal de comunicação e termina.

Arquitetura do servidor – 2



1 – instanciar, iniciar, terminar, aceitar

2 – instanciar

3 – instanciar, iniciar

4 - readObject, writeObject, fechar

5 – processAndReply

6 – métodos já definidos 7 –

instanciar, obter valores de campo

Arquitetura do servidor – 3

Tipo de dados que define o thread principal

É novo e deve ser criado. É, no entanto, quase invariável para todos os servidores. As modificações são as seguintes

- o endereço público para solicitações de serviço depende de cada servidor
- o recurso compartilhado e sua interface instanciada são específicos de cada servidor.

Tipo de dados que define o encadeamento do agente proxy de serviço

É novo e deve ser criado. É, no entanto, quase invariável para todos os servidores. As modificações são as seguintes

- se quiser que ele rode como um clone das diversas classes de clientes que acessam o servidor, deverá implementar as interfaces para cada classe relacionadas à configuração e obtenção dos atributos relevantes.

Arquitetura do servidor – 4

Tipo de dados que define a interface para o recurso compartilhado

É novo e deve ser criado. Sua organização, entretanto, é invariável para todos os servidores. A estrutura interna é a seguinte

- existe apenas um método público, alias,que decodifica o mensagem recebida (solicitação de serviço), processa-a e gera a mensagem enviada (resposta)
- a operação interna pode ser dividida em duas partes
  - validação de mensagens recebidas com eventual incorporação dos atributos do cliente
  - invocação de método e geração de mensagens de saída.

Tipo de dados que define o recurso compartilhado

Permanece praticamente inalterado. As modificações são as seguintes

- se houver referências específicas aos tipos de dados dos processos cooperantes, eles deverão ser alterados para o tipo de dados do agente proxy de serviço.

Arquitetura do servidor – 5

Tipo de dados que define o canal de comunicação

É novo e deve ser criado. Sua principal característica é encapsular as operações realizadas nos soquetes. O tipo de dados, que é fornecido no exemplos, pode ser usado tal como está ou ser modificado de acordo com requisitos específicos.

Tipo de dados que define a mensagem

É o mesmo tipo de dados usado no lado do cliente.

Arquitetura do servidor – 6

Em geral, em uma determinada aplicação estão envolvidos vários servidores, alguns deles solicitando serviços de outros. Assim, tem-se uma situação em que alguns dos servidores são simultaneamente servidores e clientes.

Isto não apresenta nenhuma dificuldade conceitual. Basta fundir ambas as funcionalidades produzindo uma arquitetura mista.

Leitura sugerida

- *Sistemas Distribuídos: Conceitos e Design, 4ª Edição, Coulouris*, Dollimore, Kindberg, Addison-Wesley
  - Capítulo 4: *Comunicação entre processos*
    Seções 4.1 a 4.4
- *Sistemas Distribuídos: Princípios e Paradigmas, 2ª Edição,* Tanenbaum, van Steen, Pearson Education Inc.
  - Capítulo 4: *Comunicação*
    Seções 4.1 a 4.3