3. Comunicação em Sistemas Distribuídos

3.1.Troca de mensagens

As mensagens são objetos de dados cuja estrutura e aplicação são definidas pelas próprias aplicações que a usarão. Sendo a troca de mensagens feita através de primitivas explicitas de comunicação:

send(destino, mensagem) envio da mensagem para o destino

receive(origem, mensagem) recebimento da mensagem enviada pela origem

As primitivas acima podem ser classificadas do seguinte modo:

a) Forma de comunicação

a.1) Direta

send: há indicação do processo receptor.

send(process, msg)

receive: há indicação do emissor.

receive(process, msg)

a.2) Indireta

send: envio para uma porta ou mailbox sem o conhecimento de qual será o receptor.

send(mailbox, msg)

receive: obtenção da mensagem guardada no mailbox, possivelmente desconhecendo a identidade do processo emissor.

receive(mailbox, msg)

b) Forma de Sincronização

b.1) Síncrono ou Bloqueante

Send: espera até que a mensagem seja recebida pelo receptor.

Receive: aguarda até a mensagem estar disponível

c) Assíncrona ou Não Bloqueante

Send: envia a mensagem mas não espera até que a mensagem seja recebida pelo receptor.

Receive: se a meensagem estiver disponível, recebe a mensagem, caso contrário continua o processamento retornando uma indicação de que a mensagem não estava disponível.

Um sistema pode possuir diferentes combinações desses tipos de primitivas, sendo que um sistema de comunicação passa a ser conhecido como síncrono se ambas as primitivas (*send, receive*), forem do tipo bloqueante. Por outro lado, um sistema de comunicação é dito assíncrono se pelo menos uma das primitivas for assíncrona.

A principal desvantagem dessa abordagem é o baixo nível de abstração que permite apenas a modelagem de troca indireta de informação entre processos.

3.2Modelo Cliente/Servidor

É um paradigma de programação que representa as interações entre os processos e as estruturas do sistema. Neste tipo de paradigma existem dois de processos:

clientes: processos que requisitam serviços;

servidores: processos que recebem requisitos , realizam uma operação e retornam serviços

Em um modelo cliente/servidor, o processo cliente necessita de um serviço (ex. Ler dados de um arquivo), então ele envia uma mensagem para o servidor e espera pela mensagem de resposta. O processo servidor, após realizar a tarefa requisitada, envia o resultado na forma de uma mensagem de resposta ao processo cliente. Note que os servidores, em um sistema deste tipo, apenas responde as requisições dos clientes,e , tipicamente, não iniciam conversação com os clientes.

A principal vantagem desta abordagem é a simplicidade, mas a implementação é menos eficiente do que a troca de mensagens pura.

Obs:

- A visão lógica da comunicação cliente/servidor possibilita a troca de mensagens
- Possibilidade de vários clientes acessarem um recurso de forma consistente e transparente através do uso de um único servidor. Mas também é possível termos ambientes com múltiplos servidores.

No modelo cliente servidor é possível a existência de vários servidores, onde os mesmos podem ser replicados, isto é, quando há várias instâncias do mesmo servidor. A replicação muitas vez ocorre por questões de desempenho, distribuição natural dos recursos ou por tolerância a falhas. Também é possível termos servidores hierárquicos, isto é, servidores que usam o(s) serviço(s) de outros servidores. Nestes casos, as localizações e conversões entre servidores deverá ser transparentes aos clientes.

3.2.1 Endereçamento

Para que um cliente envie mensagens a um servidor, primeiro o cliente, deve necessariamente conhecer o endereço do servidor, desse modo, é preciso estabelecer um esquema de identificação:

1. *um identificador único de processo quando na mesma máquina*: com um único processo por máquina basta indicar o endereço da máquina pois o kernel consegue determinar qual é o processo servidor único. Esta não é abordagem viável, pois dificilmente teremos um único processo servidor em uma máquina.
2. *Endereçamento indicando o processo e a máquina*: quando se permite mais de um processo servidor por máquina, deve−se endereçar a mensagem a esse servidor especifico. Um esquema comum é o uso de um nome composto por duas partes. Ex: 12,4 ou , sendo que 12 é a máquina e quatro é o processo. Logo, o kernel da máquina cliente usa o número 12 para enviar ao máquina correspondente e com o número 4 e o Kernel remoto determina para qual servidor a mensagem é endereçada. Nessa abordagem a identificação é incluída no código do cliente. Com isso evita−se o custo de coordenação global e evita−se ambiguidades entre processos com identificador idênticos mas em máquinas distintas. Porém, perde−se em transparência pois é preciso que o cliente conheça a localização física do servidor. Isso pode causar problemas, como por exemplo: se um servidor de banco de dados normalmente executa na maquina 12, no momento em que a mesma for desligada para manutenção, pode−se disponibilizar o mesmo serviço em outra máquina. Ao usar este esquema com maquinas fixas, o serviço poderá estar disponível em outra máquina, mas não poderá ser usado.
3. *Processos escolhem endereços que são detectados por broadcast*: Cada processo deve receber um endereço único que não envolva o número da sua máquina. Este endereço pode ser atribuído de duas formas:

- através de um escalonador de endereços de processo centralizado, que mantém um contador e ao receber uma requisição incrementa esse contador e envia o valor. A principal desvantagem é a utilização de um componente centralizado que compromete a tolerância a falhas e a escalabilidade;
- cada processo escolhe um valor aleatoriamente a partir de um espaço de endereçamento grande (ex. 64 bits) e portanto com pouca probabilidade de colisão.

O kernel emissor de uma requisição pode localizar para qual maquina enviar através do seguinte procedimento:

1) o emissor envia a mensagem para todas as maquinas (broadcast) com um pacote especial de localização contendo o endereço do processo destino;
2) em cada maquina da rede o kernel verifica se o processo está na máquina. Quem localizar envia

mensagem indicando seu endereço na rede;

3) o kernel emissor guarda essas informações para requisições futuras.

Essa abordagem tem como vantagem ser transparente mas tem um custo da mensagem broadcast.

4. *uso de servidor de nomes*: neste esquema os servidores são indicados por um identificador de alto nível (nome ASCII) que não inclui nem identificação da maquina nem do processo. Quando um cliente executa uma requisição a um servidor pela primeira vez, uma mensagem especial é enviada para um servidor de mapeamento ou servidor de nomes solicitando o número da máquina onde está o servidor.

3.2.2Primitivas confiáveis e não confiáveis de comunicação

Se as mensagens envolvidas na comunicação cliente/servidor for assumida como não confiáveis, recairá ao usuário a tarefa de controlar possíveis perdas de mensagens. Isso não é desejável uma vez que o objetivo é tornar o mais transparente possível para o usuário. Logo, é desejável que o kernel do S.O. Se preocupe em verificar se as mensagens forem corretamente recebidas garantindo dessa forma uma comunicação confiável. A figura abaixo ilustra dois protocolos para garantir a confiabilidade através do uso de mensagens especiais do tipo acknowledgment.

3.3Chamada Remota de Procedimento (RPC − Remote Procedure Call)

Chamada remota de procedimento permite que programas invoquem procedimentos ou funções localizadas em outras máquinas como se ele estivesse localmente definidos. A nível do programa, as informações são passadas do chamador para o procedimento chamado através de parâmetros e resultados são retornados através do resultado do procedimento. Deste modo, nenhuma mensagem ou entrada/saída é visível ao programador. Apesar de simples há alguns problemas nos seguintes pontos:

- chamador e chamado executam em espaços de endereçamento diferentes;
- como fazer a passagem de parâmetros e resultados quando as maquinas envolvidas têm arquiteturas distintas;
- possibilidades de falhas.

3.3.1Operação RPC Básica

A chamada de procedimento remoto deve parecer o máximo com o máximo possível com a

chamada local de procedimentos. Por isso, utiliza−se a estrutura de cliente/servidor stubs. A RPC segue os seguintes passos:

1. O procedimento chama o stub cliente de forma normal, isto é, como se fosse um procedimento local qualquer;
2. O stub cliente constrói mensagens e passa ao kernel;
3. O kernel remoto envia a mensagem ao stub servidor;
4. O kernel remoto entrega a mensagem ao stub servidor;
5. O stub servidor desempacota os parâmetros e chama o servidor;
6. O servidor realiza o trabalho solicitante e envia o resultado ao stub servidor;
7. O stub servidor empacota o resultado em uma mensagem e passa para o kernel;
8. O kernel remoto envia mensagem ao kernel cliente;
9. O kernel cliente entrega a mensagem ao stub cliente;
10. O stub desempacota o resultado e retorna ao stub cliente.

O efeito da execução de todos esses passos é a conversão da chamada local de um cliente a um stub cliente em uma chamada ao procedimento servidor sem que o cliente ou servidor percebam os passos intermediários. Via de regra os stubs são gerados automaticamente por um compilador. Essa operação básica do RPC indica alguns aspectos e problemas interessantes que serão vistos na próxima seção.

a) Passagem de Parâmetros

É preciso tratar a passagem de parâmetros e a conversão de dados. Isso é feito pela operação de "empacotamento de parâmetros" (parameter marshalling). Essa operação trata problemas de representação distinta de dados (números/caracteres)para máquinas hetergêneas:

Tanto o cliente quanto o servidor sabem os tipos de parâmetros passados (identificador do procedimento + parâmetros) e devem conseguir obter a representação correta dos dados.

Opção 1: uso de um padrão de representação de dados, como por exemplo o XDR do RPC UNIX. Essa alternativa é muito interessante por permitir a comunicação entre maquinas heterogêneas. A desvantagem é que entre máquinas homogêneas não há necessidade de incluir um custo adicional de conversão da representação da máquina para a representação padrão.

Opção 2: a mensagem é enviada no formato nativo com indicação de qual o formato utilizado. O receptor deve fazer a conversão quando for o caso. Essa opção evita custos de conversão em ambientes homogêneos, entretanto exige que clientes e servidores saibam converter qualquer formato

para a sua representação nativa.

b) Passagem de Ponteiros

Um ponteiro representa um endereço de memória que não tem nenhum significado na máquina remota.

Opção 1: proibir a passagem de ponteiros, essa obviamente não é uma alternativa desejável pelo programador;

Opção 2: Copiar o valor apontado pela variável ponteiros. As alterações são feitas remotamente e no retorno as alterações são atualizadas no chamador (semântica cópia/restauração)

Uma otimização possível consite em, sabendo que um determinado parâmetro é apenas de entrada, evita o retorno do conteúdo do ponteiro. De forma análoga, se ele for apenas de saída, ele não precisa ser copiado no envio. Esse tipo de informação pode ser fornecido pelo programador ou extraído a partir da análise do código fonte(análise estática).

c) Semântica RPC na Presença de Falhas

Na execução de uma chamda remota de procedimentos, existem cinco classes diferentes de falhas possíveis.

- O cliente não consegue localizar o servidor
- Mensagem de requisição perdida
- Mensagem de resposta perdida
- O servidor cai após receber uma requisição
- O cliente falha após enviar uma requisição (falha do cliente).

3.3.2Questões de Implementação

As questões de implementação analisadas nesta seção estão relacionadas as questões de desempenho.

a) Protocolo

Em principio qualquer mecanismo de troca de mensagem poderia dar suporte a implementação de RPC. Uma das questões a serem decididas na implementação é se o protocolo será orientado a conexão ou sem conexão. No primeiro caso, evita−se o problema de mensagens perdidas, pois isso é

tratado em baixo nível, porém, o desempenho é menor. No segundo caso, o desempenho é melhor, sendo indicado para implementação em LANs que não são confiáveis do que WANs.

b**)Confirmação (*Acknowledgment*)**

As mensagens de confirmação são usadas para detecção de falhas. No momento de implementar, é preciso decidir se os pacotes serão confirmados individualmente ou não. Assim temos dois tipos de protocolos: *stop−and−wait* (no momento em que um pacote for danificado ou perdido, o cliente não recebendo o ACK providência o reenvio), e o blast (o servidor envia mensagem ACK apenas no final).

No caso do protocolo *Blast*, há duas formas de tratar o fato de apenas alguns pacotes terem sido entregues e outros não. Uma hipótese é ignorar todos os pacotes e solicitar a retransmissão de todos os pacotes. Outra possibilidade é realizar uma repetição seletiva (selective repeat) não compensa devido ao custo de controle envolvido.

Outro problema refere−se ao fato dos buffers de recepção serem limitados. O envio de várias mensagens consecutivas pode causar um *overrum error*. O *overrum error* caracteriza−se pela incapacidade do receptor de receber um pacote que chega corretamente causando dessa forma uma perda de mensagem. Na prática esse é um erro mais sério do que a perda por ruído ou outros problemas físicos na rede. Com o uso de *stop−and−wait* em princípio esse erro não ocorre pois uma mensagem é enviada de cada vez. No entanto, a mesma, poderia ocorrer caso diferentes emissores enviarem várias mensagens ao mesmo tempo.

3.4Comunicação de Grupo

Há vária formas aplicações para as quais tem−se comunicação com múltiplos processadores. Ex.: grupos de servidores de arquivos que fornecem serviço de arquivos tolerantes a falhas. Um processo pode enviar mensagens para um grupo de servidores sem saber quantos eles são ou onde localizam−se, sendo que tais parâmetros podem mudar na próxima chamada.

Um grupo é uma coleção de processos que agem juntos. Quanto uma mensagem é enviada ao grupo, todos os membros recebem a mesma mensagem.

Grupos são dinâmicos: podem ser criados e destruídos, processos podem entrar ou abandonar um grupo; e um mesmo grupo pode pertencer a mais de um grupo.