UMA IMPLEMENTAÇÃO DA CHAMADA REMOTA DE PROCEDIMENTO

CELIO ESTEVAN MORON * ARTHUR JOÃO CATTO **

SUMARIO

Este trabalho descreve a implementação do mecanismo CHAMADA REMOTA DE PROCEDIMENTO usada em Sistemas Distribuídos para realizar operações remotas. E discutida a troca de mensagem sobre a qual a Chamada Remota de Procedimento é construída, bem como os requisitos necessários à confiabilidade.

ABSTRACT

This work describes the implementation of the REMOTE PROCEDURE CALL mechanism used in Distributed Systems to perform remote operations. Message passing, which is the base for Remote Procedure Call mechanism, is dicussed as well as the reliability requirements.

*Bacharel em Ciência da Computação (UFSCar,1979), Mestre em Ciência da Computação (ICMSC,USP,1986); Redes de Computadores, Sistemas Distribuídos, Sistemas Operacionais; Universidade Federal de São Carlos, (0162) 71-1100 Ramal 143.

**Engenheiro Civil (EESC,USP,1970), Ph.D. in Computer Science (University of Manchester, 1981); computação paralela, arquiteturas dirigidas pelo fluxo de dados, técnicas de programação; Centro Tecnológico para Informática - CTI (0192) 42-1000 Ramal 176.

1. INTRODUÇÃO

Processos concorrentes podem se comunicar e sincronizar de acordo com a necessidade. A comunicação entre processos concorrentes permite que um processo influencie a execução de outro. Muitas vezes quando dois processos se comunicam é preciso que haja sincronização para que a ordem de ocorrência de eventos seja estabelecida.

A comunicação é feita através de uma ação que é detectada pelo outro processo. Esta ação pode ser a alteração do valor de alguma variável ou o envio de alguma mensagem. No primeiro caso dizemos que a comunicação é feita por meio de **variáveis partilhadas** e no segundo caso por meio da **troca de mensagens**.

A comunicação entre processos por meio de variáveis partilhadas adapta-se a sistemas de computação com memória comum, como sistemas multiprogramados e multiprocessadores.Regiões críticas, monitores e semáforos são exemplos de mecanismos de comunicação de processos baseados no uso de variáveis partilhadas.

Por outro lado em um sistema distribuído a comunicação entre processos tem que ser por troca de mensagens, uma vez que o sistema não dispõe de memória partilhada.

Os sistemas distribuídos podem ser de duas categorias dependendo de como implementam e usam a noção de processo e

sincronização. Estas duas categorias são denominadas sistemas orientados por mensagem e sitemas orientados por procedimento [1].

A comunicação em um sistema orientado por mensagem é uma extensão daquela feita por meio de variáveis partilhadas para um ambiente distribuído, onde em vez da comunicação ocorrer por meio da memória, ela é feita através de um sistema de comunicação. Este tipo de sistema tem a vantagem de permitir implementação simples e eficiente, e introduzir diretamente o paralelismo [2]. Por outro lado, este tipo de sistema tem diversas desvantagens do ponto de vista do projeto da linguagem. Temos a introdução de uma primitiva cujo controle é muito diferente do mecanismo de controle dos procedimentos. Isto não é desejável em linguagens baseadas em procedimentos, como Algol ou Pascal, onde a troca de mensagens requer uma nova primitiva e não simplesmente uma extensão da chamada local de procedimento. Outra desvantagem é a introdução de inconsistências dentro da linguagem, na estrutura de verificação da compatibilidade de tipos.

A comunicação em um sistema orientado por procedimento é uma extensão da chamada de procedimento para um sistema distribuído. Sua maior vantagem é a padronização das chamadas locais de procedimentos e chamadas externas de procedimentos. O programador pode não perceber que vários processos do seu programa são executados em diferentes processadores operadores, ou seja, isto pode ficar transparente ao programador. Desta forma o programador não tem que sair do seu ambiente usual de

programação para se preocupar com detalhes de troca de mensagens, e tudo se passa como se estivesse trabalhando com um sistema centralizado. Por outro lado o maior problema deste tipo de sistema está na propriedade que os procedimentos têm de sempre retornarem ao ponto da chamada. Garantir que uma chamada externa de procedimento sempre retorne adequadamente é extremamente difícil em um sistema distribuído, onde um processador ou o sistema de comunicação podem falhar. Por causa do exposto tem sido difícil a extensão de chamadas locais para remotas, sem a introdução de algum mecanismo deselegante para controle do tempo de resposta, códigos de erros ou armadilhas para detecção de falhas.

2. SISTEMAS ORIENTADOS POR MENSAGEM

Esta classe de sistemas possui primitivas para enviar e receber mensagens. Uma **mensagem** é uma estrutura de dados usada para enviar informações de um processador operador para outro. Um **nome** é um identificador pelo qual uma particular mensagem pode ser reconhecida.

Uma mensagem pode ser enviada de um processo para outro pela execução da seguinte primitiva:

ENVIA nome

PARA destino

A mensagem contém o valor da estrutura de dados **nome** no

momento em que a primitiva ENVIA é executada. O **destino** indica para qual processo esta mensagem deve ser enviada.

Uma mensagem pode ser recebida por um processo pela execução da seguinte primitiva:

RECEBE nome

DE origem

Esta primitiva atribui à estrutura de dados **nome** o valor da mensagem recebida. A **origem** dá ao programador controle sobre os pontos de onde a mensagem pode provir.

Chamaremos o processo que executa a primitiva ENVIA de **processo emissor**, e o processo que executa o RECEBE correspondente de **processo receptor**.

2.1. SINCRONIZAÇÃO

Uma primitiva de sincronização ENVIA pode causar um atraso no seu processo emissor até que o RECEBE correspondente seja executado pelo processo receptor. Uma primitiva é dita **não bloqueada** se sua execução nunca atrasa o processo emissor e **bloqueada** caso possa atrasar o seu processo emissor [3].

Numa primitiva não bloqueda, as mensagens são colocadas em um buffer durante o intervalo entre o envio e o recebimento. Se o buffer estiver cheio quando um ENVIA é executado, há duas

opções: o ENVIA pode atrasar o processo emissor até que exista lugar no buffer para a mensagem, ou o ENVIA pode retornar um código para o processo emissor, indicando que, o buffer está cheio e a mensagem não pode ser enviada. Da mesma forma, um RECEBE pode atrasar o processo receptor até que uma mensagem esteja disponível ou então retornar um código para o processo receptor, indicando que nenhuma mensagem está disponível.

A troca de mensagens usando buffer, permite que o processo emissor envie uma mensagem antes que o RECEBE correspondente seja executado. Este modo de troca de mensagens é também chamado **assíncrono**.

A troca de mensagens sem usar buffer faz com que a execução de um ENVIA atrase o processo emissor até que o RECEBE correspondente seja executado. Só então a mensagem é transferida e processada. Este modo de troca de mensagens é também chamado **síncrono**. Na troca síncrona de mensagens a transferência da mensagem é o ponto de sincronização entre o processo emissor e o receptor.

Um RECEBE bloqueado implementa implicitamente a sincronização entre o emissor e o receptor, uma vez que o RECEBE espera até que a mensagem seja enviada. Um comando de troca de mensagem bloqueado pode ter o mesmo efeito semântico que o não bloqueado correspondente, pelo uso da **comunicação seletiva** que é baseada no comando com guarda de Dijkstra [4].

Em um comando de comunicação seletiva, comandos com guarda

têm a seguinte forma:

guarda --> lista de comandos

O guarda consiste de uma expressão booleana, opcionalmente seguida por um comando de troca de mensagem. O guarda obtém êxito se a expressão booleana for verdadeira e a execução do comando de troca de mensagem não causar atraso; o guarda falha se a expressão booleana for falsa; o guarda nem falha nem obtém êxito se a expressão booleana for verdadeira mas o comando de troca de mensagem não puder ser executado sem causar atraso. Quando uma lista de comandos com guarda é selecionada para execução, seus comandos são executados sucessivamente da esquerda para a direita.

Comandos com guarda permitem a construção de comandos alternativos e repetitivos com comportamento não-determinístico.

O comando alternativo tem a seguinte forma:

```
IF  G1 --> S1
/   G2 --> S2
    ...
/   Gn --> Sn
end
```

Quando um comando alternativo é alcançado, todos os seus guardas são avaliados simultaneamente. Se ao menos um guarda obtém êxito, um deles é selecionado nao determinísticamente; o comando de passagem de mensagem correspondente é executado (se

presente) e a lista de comandos que segue o guarda é executada. Se todos os guardas falharem o comando aborta. Se todos os guardas não falham nem obtêm êxito, a execução é atrasada até que algum guarda tenha êxito.

O comando repetitivo tem a seguinte forma:

```
DO  G1 --> S1
/   G2 --> S2
    ...
/   Gn --> Sn
end
```

A execução do comando repetitivo é semelhante à do comando alternativo, sendo que a seleção e execução de um comando com guarda é repetida até que todos os guardas falhem, com o tempo o comando repetitivo termina em vez de abortar.

3. SISTEMAS ORIENTADOS POR PROCEDIMENTO

Em um sistema orientado por procedimento temos uma primitiva de comunicação e sincronização de processos denominada chamada remota de procedimento. Os processos cooperam entre si, enviando e recebendo mensagens através do Sistema de Comunicação, sendo que estas mensagens são chamadas para procedimentos externos, e resultados da execução de procedimentos externos. Durante a execução de uma chamada remota de procedimento, o processo que solicita a operação é denominado **cliente**, enquanto que o processo que executa é denominado

servidor. O nome da primitiva deriva do fato de um processo cliente "chamar" um procedimento que é executado em uma máquina remota, por um processo servidor.

O processo cliente e o processo servidor executam duas trocas de mensagem, da seguinte maneira: o processo cliente ENVIA a chamada e, em seguida, RECEBE os resultados, enquanto o processo servidor RECEBE a chamada e, depois, envia os resultados.

Do ponto de vista do processo cliente a chamada remota de procedimento é executada do seguinte modo: a identificação do procedimento e os valores dos argumentos de entrada são enviados para o servidor apropriado e o processo cliente é atrasado, até que os resultados tenham sido recebidos e atribuídos aos argumentos de saída correspondentes.

O processo servidor, por sua vez, recebe uma chamada para um procedimento, executa os comandos correspondentes e envia os resultados para o processo cliente.

Uma chamada remota de procedimento só termina depois que o cliente receber o resultado desejado. Uma chamada remota de procedimento é uma extensão natural de uma chamada de procedimento em uma linguagem sequencial. Numa linguagem sequencial nós temos a transferência do controle para sub-programas em um mesmo processador operador, enquanto na chamada remota de procedimento temos a transferência do controle para

sub-programas em processadores operadores diferentes.

Numa chamada remota de procedimento o processo cliente faz uma requisição para execução de um procedimento externo e fica bloqueado esperando o resultado. Como o cliente e o servidor são executados em processadores operadores diferentes pode acontecer uma falha no sistema de comunicação ou em um processador operador. Desta forma o cliente pode ficar esperando um resultado indefinidamente. Podemos ter ainda, por exemplo, o caso de um processo A que é suspenso à espera de que uma condição torne-se verdadeira e esta condição só pode ser mudada por uma chamada de um processo B. Se o processo B por sua vez for suspenso à espera de que uma condição torne-se verdadeira e esta condição só pode ser mudada por uma chamada do processo A, um processo ficará esperando pelo outro indefinidamente. Quando isto acontece dizemos que ocorreu um intertravamento dos processos ou deadlock.

A confiabilidade não pode ser obtida sem o uso de mecanismos deselegantes que, passado um tempo estipulado, tomam providências caso uma das situações expostas acima aconteça [5], [6], [7].

4. PRIMITIVA ENVIA

Uma mensagem é enviada de um processo para outro através de operações de saída usando uma porta de entrada/saída serial.

A primitiva bloqueada ENVIA, também chamada de ENVIA

SINCRONO, tem a seguinte estrutura:

ENVIA (processo-destino, mensagem, VAR estado)

O processo emissor não pode prosseguir até que o processo receptor confirme ter recebido a mensagem. O parâmetro "processo-destino" é o identificador do procesador operador para onde a mensagem deve ser enviada, o parâmetro "mensagem" é um conjunto de bytes que constituem a informação que deve ser transmitida e "estado" retorna um valor booleano que, no caso de ser verdadeiro, significa que a mensagem foi enviada e, caso contrário, que a operação não obteve sucesso mesmo depois de várias tentativas. Neste último caso devem ser enviadas mensagens alertando sobre o mau funcionamento do sistema.

5. PRIMITIVA RECEBE

A mensagem é recebida de outro processo através de operações de entrada usando uma porta de entrada/saída serial.

A primitiva bloqueada RECEBE, também chamada de RECEBE SINCRONO, tem a seguinte estrutura:

RECEBE (VAR mensagem, VAR estado, processo-fonte, tempo-resposta)

O processo receptor não pode prosseguir até que o processo emissor envie uma mensagem ou até que o tempo-resposta seja esgotado. O parâmetro "mensagem" é um conjunto de bytes que constituem a informação recebida e "estado" retorna um valor

booleano que, no caso de ser verdadeiro, significa que a mensagem foi recebida do "processo-fonte" e, caso contrário, que passado o "tempo-resposta" nenhuma mensagem foi recebida.

6. PRIMITIVA CHAMADA REMOTA DE PROCEDIMENTO

A primitiva CHAMADA REMOTA DE PROCEDIMENTO (CRP) é ativada durante a fase de EXECUÇÃO quando o processo faz uma referência a um procedimento externo. A implementação de uma primitiva CRP envolve o envio pelo cliente de uma requisição com uma mensagem para o servidor apropriado, em seguida à recepção da resposta do servidor. O servidor recebe o pedido de execução de um procedimento, executa o trabalho e envia o resultado para o cliente.

A CHAMADA REMOTA DE PROCEDIMENTO tem a seguinte estrutura:

```
CHAMADA   REMOTA   (servidor,   cliente,   procedimento,
                   parâmetros-entrada,   VAR resultado,
                   VAR estadoCRP, tempo-resposta)
```

O "servidor" especifica o processador operador que atende a CRP, "procedimento" é a especificação do procedimento a ser executado, "parâmetros-entrada" são os parâmetros pedidos pelo procedimento que vai ser executado remotamente, "resultado" recebe os parâmetros resultados enviados pelo servidor, "estadoCRP" retorna o estado da CRP e "tempo-resposta" especifica o tempo que o cliente poderá esperar por uma resposta. O "estadoCRP" tem, para a chamada remota de

procedimento, o seguinte significado:

estadoCRP = executado: O procedimento especificado foi executado pelo servidor e as respostas estão no "resultado".

estadoCRP = não recebido: Não existe o processador operador especificado ou este processador operador está com defeito ou o Sistema de Comunicação está com defeito.

estadoCRP = não executado: O processador operador aceitou a chamada mas o tempo-resposta expirou e não chegou o resultado da execução remota.

O formato da mensagem na chamada do procedimento remoto e no retorno do resultado está especificado na figura 1.

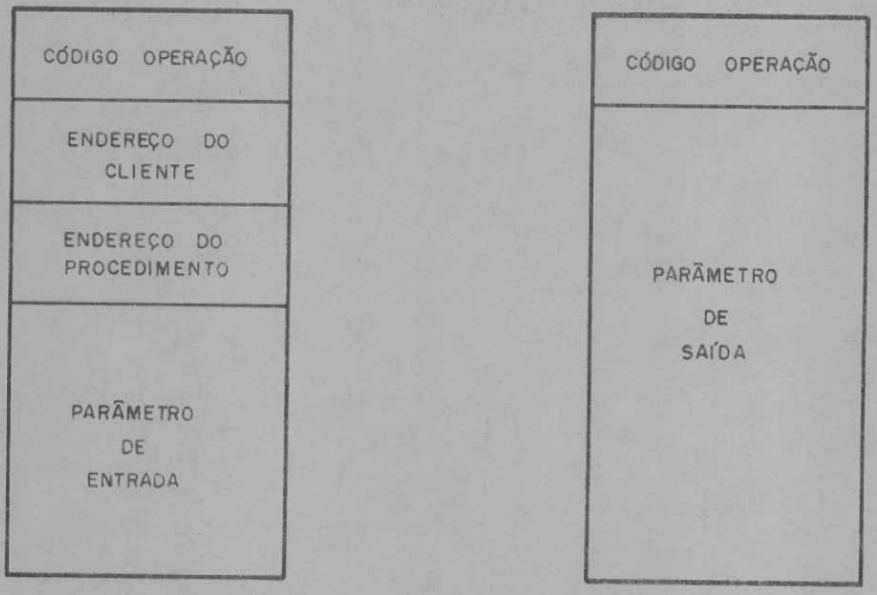

Figura 1 Formato da mensagem na CRP

O código de operação tem o seguinte significado:

Código operação = CRP: Significa que esta mensagem é uma chamada remota de procedimento.

Código Operação = RES: Significa que esta mensagem é o resultado de uma chamada remota de procedimento.

7. CONCLUSÃO

A Chamada Remota de Procedimento é um mecanismo muito poderoso em programação distribuída porque pode ser transparente e dar ao programador a impressão de que está usando um sistema centralizado e não um sistema distribuído. O programador portanto se abstrai de todo detalhe da implementação e comunicação entre os processadores.

A Chamada Remota de Procedimento na forma discutida neste trabalho foi implementada [8] usando sistemas de computação com central de processamento Z-80A, memória principal com 64 bytes, interface para impressora, terminal de vídeo e unidades de disco flexível. A comunicação entre processadores foi feita através de um Sistema de Comunicação utilizando-se interface serial.

8. REFERENCIAS BIBLIOGRAFICAS

[1] LAUER, H. C. e NEEDHAM, R. M. - "On the Duality of Operating Systems Structures" - Poceedings of the 2nd International Symposium on Operating Systems, IRIA, (Oct 1978), ou em Operating Systems Review 13, 2 (April 1979), pg 3-19.

[2] NELSON, B. J. - "Remote Procedure Call" - Dissertação Doutorado - Departament of Computer Science, Carnegie-Mellon University.

[3] ANDREWS, G. R. e SCHNEIDER, F. B. - "Concepts and Notations for Concurrents Programming" - ACM Comp. Surv, 15, 1, (March 1983), pg 3-43.

[4] DIJKSTRA, E. W. - "Guarded Commands, Nondeterminancy, and Formal Derivation of Programs" - Commun. ACM, 18, 8, (Aug 1975), pg 453-457.

[5] MAROVAC, N. - "On Interprocess Interaction in Distributed Architectures" - Computer Architecture News 11, 4, (Sept 1983), pg 17-22.

[6] SPECTOR, A. Z. - "Performing Remote Operations Efficiently on a Local Computer Network". Commun. ACM, 25, 4, (April 1982) pg 246-260.

[7] SHRIVASTARA, S. K. e PANZIERE, F. - "The Design of a Reliable Remote Calls Mechanism", IEEE, Transactions on Computer, July 1982

[8] MORON, C. E. - "Núcleo Básico para um Sistema Distribuído de Tempo Real" - Dissertação de Mestrado ICMSC, USP, 1986.