RPC e Exclusão mútua

Trabalho 1 – Programação Paralela e Distribuída

Fernando Zatt Grando
Estudante de Graduação em Engenharia de Computação
Pontifícia Universidade Católica do Rio Grande do Sul
Porto Alegre, Brasil
fernando.grando@acad.pucrs.br

Thomas Volpato de Oliveira
Estudante de Graduação em Engenharia de Computação
Pontifícia Universidade Católica do Rio Grande do Sul
Porto Alegre, Brasil
thomas.volpato@acad.pucrs.br

***Abstract*—The present work presents the implementation of a distributed algorithm using RPC to provide access to critical section between processes in machines running Linux.**

Keywords—RPC; distributed algorithm; critical section; Linux;

I. INTRODUÇÃO

Algumas vezes as vantagens de um sistema distribuído, como redundância e poder de processamento, podem deixar de ser atrativas, dada a complexidade no desenvolvimento de software para tais sistemas. O Remote Procedure Call – RPC – é uma forma de comunicação entre processos que permite a execução de sub-rotinas remotamente em qualquer máquina que disponibilize tal serviço [1] de forma simples. O RPC, em uma forma de uso padrão, abstrai toda a parte de comunicação do programador, requerendo somente a implementação dos procedimentos que serão disponibilizados remotamente. Nesse trabalho, feito em linguagem C, o RPC é utilizado na programação do algoritmo de acesso a Seção Crítica (SC) distribuída.

II. FUNCIONAMENTO

O algoritmo implementado consiste em cada máquina pedir acesso a todas as outras no ambiente antes de fazer o acesso a SC. Para exemplificar, vamos supor um cenário com três máquinas iguais e com relógios sincronizados, A, B e C. Caso a máquina A queira acessar a SC ela mandará uma requisição para as máquinas B e C. A máquina B não faz uso da SC e prontamente permite o acesso. Entretanto a máquina C quer acessar a SC também, então ela verifica se o *timesamp*[1] da máquina A é menor que o seu próprio: caso seja menor, permite, pois a máquina A não permitirá seu acesso (C tem *timestamp* maior); caso seja maior, coloca o pedido de A em uma fila, enviando a permissão mais tarde, assim que terminar de acessar a SC.

[1] O *timestamp* é um valor discreto do tempo em um dado instante, computado utilizando as últimas 6 casas retornadas pela função *time()* concatenadas com o valor em milissegundos retornado pela função *gettimeofday()*, ambas existentes na linguagem C.

III. INSTALAÇÃO

Para que os procedimentos remotos fiquem disponíveis para outras máquinas é necessário registrá-los na máquina local, que precisa dispor do serviço *rpcbind*[2]. É necessário também que o ambiente de máquinas compartilhe um diretório comum no sistema de arquivos (no trabalho é o Drive-H). Nesse diretório existirá o arquivo de configuração do ambiente (*env.conf*), o arquivo com uma semente randômica (*random.txt*), e o arquivo, que representa a SC, onde o resultado dos cálculos é armazenado (*calculo.txt*).

Tendo o serviço de RPC adequadamente funcionando, é necessário ajustar os valores do arquivo *env.conf*, informando a localização do arquivo de SC e todos os IPv4 das máquinas participantes. Esse arquivo será carregado por todas as máquinas para informar quem são os possíveis vizinhos, bem como o número de permissões necessárias para acessar a SC. Também é atribuída a cada uma das máquinas um nome e uma operação matemática. Esse trabalho suporta operações de adição, subtração, multiplicação, divisão e potenciação sobre o último valor presente no arquivo calculo.txt. É esperado que, antes da execução do programa, o arquivo calculo.txt contenha na última linha um valor numérico para ser usado.

IV. IMPLEMENTAÇÃO

O trabalho foi desenvolvido em ambiente Linux e implementado na a linguagem C utilizando como compilador o *gcc* versão 4.7.2 e o *rpcgen* para gerar os arquivos da estrutura RPC. As bibliotecas *lm* e *lpthread* também são utilizadas para as funções de potenciação e criação de threads, respectivamente. O programa implementado apresenta funções de cliente e servidor durante a operação na máquina host, então, para facilitar a compreensão, a partir de agora o chamaremos de entidade.

Na descrição fornecida para o *rpcgen* no arquivo *prog.x* existe uma *sctruct* e três procedimentos para a execução remota. Na máquina host, o serviço é registrado como CALCULATOR com ID para usuário 0x20000001 [2]. Os procedimentos descritos são:

[2] O nome do pacote, bem como suas dependências, pode variar de acordo com a distribuição Linux utilizada.

- *start_calculations(long int r)*: inicia na entidade a *thread* cliente, que recebe um número randômico para gerar valores aleatórios de espera e número de operações na SC.
- *ok(struct machine m)*: utilizado para dar permissão a entidade host do procedimento. A entidade utiliza o valor do IPv4 da máquina que autorizou o acesso.
- *request(struct machine m)*: utilizado para pedir permissão a entidade host do procedimento. A entidade utiliza os valores de *timestamp* e IPv4 da máquina que está requerendo acesso.

A *struct machine* que é utilizada na comunicação entre entidades contém os campos:

- **ipv4**: *string* com o IPv4 da entidade (deve ser único).
- **name**: nome da máquina.
- **operation**: operação matemática que a entidade faz.
- **timestamp**: valor do *timestamp*, quando necessário.

Cada instância da entidade possui uma *thread* responsável por gerenciar sua atividade de acesso a seção crítica, que consiste basicamente em um loop, enquanto existirem operações a realizar:

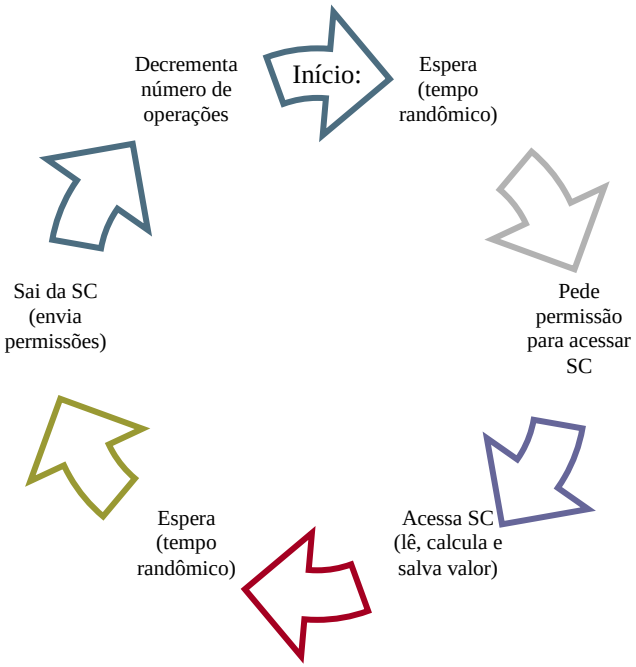

Figura 1 - Estados da *thread* cliente.

Foi necessário adicionar localmente dois sistemas de *mutex* para acessar as listas de permissões e de pendências, para garantir a consistência dos dados, já que o cliente pode acessar juntamente com o servidor da entidade.

A cada acesso feito a entidade, seja para enviar uma permissão ou para pedir uma requisição, uma nova *thread* é criada para tratar a chamada.

Os comandos para compilar o cliente (inicia a entidade) e o servidor (a entidade) são:

- *gcc defines.h configs.h prog.h file_manip.h file_manip.c prog_xdr.c prog_svc.c prog_clnt.c prog_server.c -o prog_server –lm -lpthread*
- *gcc defines.h configs.h prog.h file_manip.h file_manip.c prog_xdr.c prog_clnt.c prog_client.c -o prog_client*

V. Execução

Para iniciar as entidades deve-se executar o *prog_server* em cada máquina que teve o IPv4 cadastrado no arquivo *env.conf*. Uma execução com sucesso mantém o terminal aparentemente "travado", aguardando pela inicialização da *thread* cliente na entidade. Caso contrário, se alguma mensagem for retornada, ou ocorrer algum erro na execução do *prog_server*, pode ser necessário reiniciar alguns serviços:

- *sudo -i service portmap stop*
- *sudo -i rpcbind -i -w*
- *sudo -i service portmap start*

Com o *prog_server* executando em todas as máquinas é necessário executar, somente em uma delas, o *prog_client*. Ele será o responsável por iniciar a *thread* cliente em cada uma das entidades existentes no ambiente.

Assim que o *proc_client* é executado, a atividade da entidade pode ser vista diretamente no terminal onde foi executado o *prog_server*.

VI. Resultados

O programa foi testado em dois ambientes, um virtualizado com três máquinas e um real com cinco máquinas. Em ambos ambientes o algoritmo funcionou propriamente, permitindo um acesso mutualmente exclusivo a SC (escrita coerente no arquivo, por ordem de *timestamp* e com operação matemática correta).

```
Usando seed: 914447749
Carregando vizinhos...
        [A] : [10.32.148.28] (subtracao)
 <#>    [B] : [10.32.148.26] (adicao)
        [C] : [10.32.148.27] (multiplicacao)
        [D] : [10.32.148.31] (divisao)
        [E] : [10.32.148.39] (potenciacao)

Testando timestamp: 390329
Criando thread cliente (0): OK

Operacoes restantes: 5 {3}...
Aguardando permissão:

[ meu timestamp = 390858 ]
```

Figura 2 – Inicialização da entidade. O arquivo com a descrição do ambiente é carregado e a *thread* cliente começa a executar.

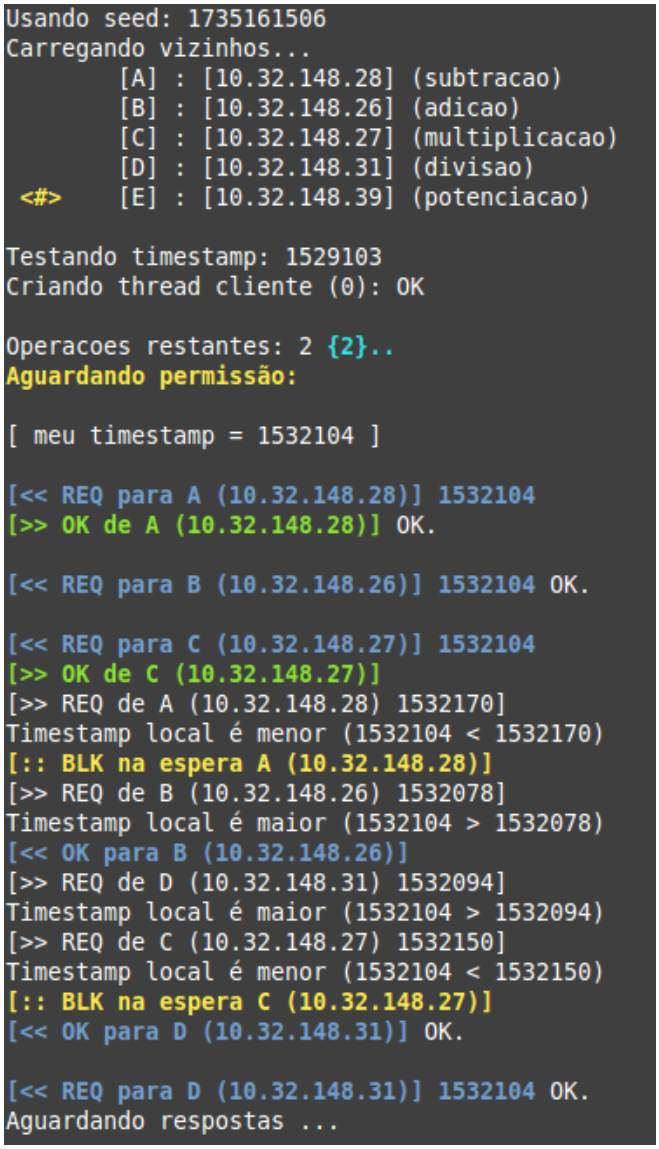

Figura 3 - Entidade executando. É possível observar os pedidos (<<REQ) e as permissões (<<OK) enviados, bem como os pedidos armazenados (::BLK) e permissões recebidas (>>OK).

Figura 4 - Acesso a seção crítica. Após realizar processamento na SC, a entidade envia as permissões para os demais aguardando na fila.

VII. LIMITAÇÕES

Pela natureza do algoritmo é possível que para um grande número de máquinas a rede sofra um impacto considerável, visto que há sempre a necessidade da todos os nodos comunicarem-se para acessar a SC, mas esse é um preço, relativamente baixo, ao se comparar a simplicidade e o benefício proporcionado com relação ao acesso mutualmente exclusivo de forma distribuída.

Com relação específica ao programa desenvolvido, entre as principais limitações estão:

- sistema operacional: funciona corretamente no Linux (não testado em outras plataformas).
- totalmente dependente do serviços de *rpcbind* da máquina: por padrão, o usuário comum não é autorizado a acessar.
- devido a forma de cálculo do *timestamp*, o programa pode rodar no máximo 11,56 dias gerando *timestamps* subsequentes. A partir desse

valor, o valor do próximo *timestamp* começa do zero novamente.

- é necessária uma pasta compartilhada comum: o usuário comum, por padrão, não é autorizado a fazer *mount* de partições remotas.
- a principal semente randômica é provida pelo arquivo random.txt.
- a função que captura o IP da máquina local utiliza o IPv4 da primeira interface disponível no comando *ifconfig*. Essa deve ser, portanto a interface que se comunica com as demais máquinas do ambiente.
- o ambiente é limitado ao que existe no arquivo *env.conf*: não é possível adicionar ou remover máquinas dinamicamente, nem modificar entidades que já estejam executando.

VIII. REFERÊNCIAS

[1] RPC - http://en.wikipedia.org/wiki/Remote_procedure_call

[2] ID - http://tools.ietf.org/html/rfc1831.html#section-7.3