Comunicação entre Processos Distribuídos

Acesso à sessão crítica utilizando o algoritmo baseado na passagem de token

Marcelo Melo Linck
Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS
Faculdade de Informática - FACIN
Porto Alegre, Brasil
marcelo.linck@acad.pucrs.br

***Resumo*—Este documento relata as dificuldades, problemas, soluções e resultados encontrados durante o desenvolvimento de um Sistema Distribuído com acesso à uma sessão crítica baseado no algoritmo de passagem de Token.**

***Abstract*—This paper relates the difficulties, problems, solutions and results found during the development of a Token based algorithm of a Distributed System accessing a Critic Session.**

***Palavras Chave*—Token; Sessão Crítica; Sistema Distribuido; RPC; Remote Procedure Call.**

I. INTRODUÇÃO

Este documento apresenta o fluxo de desenvolvimento de um Sistema Distribuído com acesso a uma Sessão Crítica utilizando o algoritmo de passagem de Token. É mostrado na parte de desenvolvimento a estrutura do código, a ideia aplicada, a estrutura do sistema, e a forma de compilação. Além disso são mostrados os resultados e o tratamento contra a perda do Token, desenvolvido em uma segunda etapa do projeto.

II. PROJETO E DESENVOLVIMENTO

A. *Desafio inicial*

O desafio apresentado foi o desenvolvimento de um Sistema Distribuído no qual houvesse acesso a uma sessão crítica (arquivo) em uma rede compartilhada entre os processos. A possibilidade de acesso a escrita deste arquivo seria estabelecida pela aquisição de um Token, o qual passaria de servidor a servidor, formando um anel. Aquele que possuísse o Token, teria o acesso. Este sistema foi desenvolvido em C, utilizando a estrutura RPC (Remote Procedure Call) da *Sun*™.

B. *Estrutura do Sistema*

O sistema inicial é montado utilizando cinco máquinas, cada máquina possui o seu próprio servidor e o seu cliente. Estas somente podem comunicar-se com a máquina definida como sua máquina seguinte, e essa comunicação é feita somente para realizar a passagem do Token. Para a inicialização do sistema, foi implementado um cliente proprietário somente de uma máquina, o qual inicializa cada um dos processos. Um diagrama desta estrutura é apresentado na fig.1 a baixo.

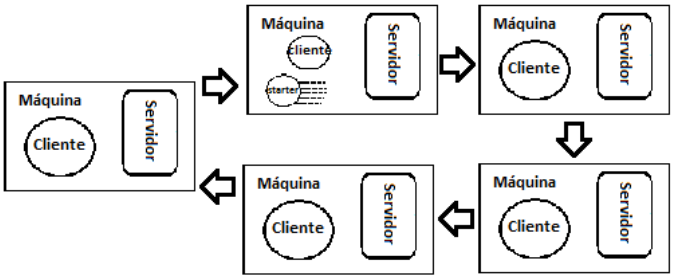


Fig. 1 – Estrutura do Sistema

C. *Idéia aplicada*

Para evitar o acesso à Sessão Crítica ao mesmo tempo por mais de um servidor, o algoritmo de Token foi implementado. Este, seria uma mensagem (Token) que passaria de processo à processo. Quando um cliente necessita acesso ao arquivo texto, ele envia uma mensagem ao seu servidor fazendo a solicitação, se o mesmo possui o Token, ele envia um *O.K.* de volta, permitindo o acesso do cliente, caso ele não possua, ele fica constantemente verificando a chegada do Token, e enquanto isso, o cliente espera o seu acesso ser concedido.

Para o desenvolvimento da verificação da existência do Token, e a necessidade de usá-lo pelo cliente, foi criada uma thread, onde é realizada esta tarefa. O cliente é um conjunto de funções, onde cada máquina possui sua ID, que define a operação a ser feita no arquivo texto, o qual é acessado em um tempo aleatório de 1 a 15 segundos.

D. *Estrutura interna*

O Servidor e o cliente se comunicam através de duas funções chamadas *writerequest* e *writerelease*. A primeira, solicita o acesso ao servidor, onde o mesmo, retorna 0 se possível, e 1 caso não seja possível o acesso; a segunda função, o cliente chama ao fim de seu acesso, informando que a tarefa terminou, e que o Token não é mais necessário naquele momento.

Para a passagem do Token, existe uma função chamada *passtoken*, ela é chamada de um servidor para o outro, e sua

única funcionalidade é configurar a variável *hasToken* de 0 para 1 no próximo servidor. Além disso, como informado na sessão *II.C.*, o servidor possui uma thread que é executada paralelamente com suas funções, pode-se dizer que ela seria somente para a verificação do Token, enquanto o programa paralelo, seria para comunicação com o cliente, e o recebimento da mensagem Token.

Por fim, existe o cliente externo chamado *Starter*, este é chamado para inicializar o sistema, onde ele faz a conexão e comunicação com cada servidor cadastrado, chamando a função *start*. Esta função configura as variáveis de cada servidor e inicializa a thread de verificação.

Toda a estrutura interna de uma máquina pode ser verificada na figura abaixo. Onde além disso, são mostradas as ligações entre processos, nomeadas de acordo com o código.

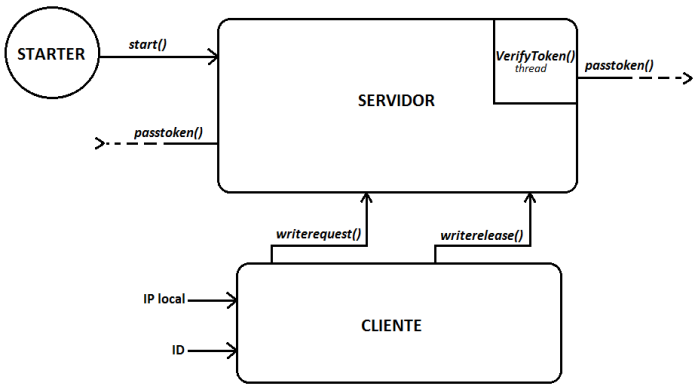


Fig. 2 - Estrutura interna

E. Compilação e Execução

Incialmente, é necessário a configuração de cada servidor separadamente antes da compilação, ou seja, cada servidor deve ter seu IP local e o IP do seu próximo configurados na função *start* antes que o programa possa ser compilado. Para o cliente, não é necessário haver uma diferença entre os arquivos, o mesmo executável do cliente pode ser utilizado em todas as máquinas. E como o *Starter* somente será rodado de uma máquina, ele somente precisa ser configurado uma vez, porém, assim como no servidor, este arquivo também deve ter os IP informados para cada conexão antes da compilação.

Após execução dos passos indicados acima, o passo-a-passo (deve ser rodado em um terminal Linux) abaixo auxilia na geração do RPC, compilação e execução de cada processo.

1) Geração do RPC

O código abaixo gera os arquivos de RPC, ou seja, o stub do cliente (*token_clnt.c*) e do servidor (*token_svc.c*), os quais guardam as funções de comunicação, e a biblioteca *token.h* com a declaração das mesmas.

```
$ rpcgen token.x
```

2) Compilação

Para a compilação, cada servidor terá o seu executável[1], e todos são compilados da mesma maneira. O executável do cliente[2] pode ser distribuído, através de uma pasta compartilhada (recomendável) para todas as máquinas. E o executável do *Starter* pode ser mantido local.

```
$ gcc –pthread token_server.c token_svc.c token_clnt.c –o server
$ gcc token_clnt.c token_svc.c –o client
$ gcc token_clnt.c starter.c –o starter
```

3) Execução

Na execução, excepcionalmente uma máquina irá seguir a sequência abaixo (*a*), enquanto as outras podem seguir a sequência *b*.

a) Sequência:

```
$ ./server
$ ./starter[3]
$ ./client <IPlocal> <ID>
```

b) Sequência:

```
$ ./server
$ ./client <IPlocal> <ID>
```

A tabela abaixo informa a relação da ID de cada cliente e a operação feita no arquivo texto.

TABELA I - Relação entre IDs e operações

ID	Operação
0	+10
1	-5
2	x3
3	+2
4	-4

III. RESULTADOS

Nessa seção, são apresentados os resultados de um teste do sistema feito com apenas três máquinas. Estes resultados são mostrados através de imagens da tela com o resultado do chamamento de cada função nos processos, e o arquivo após cinco execuções de cada um dos 3 clientes (IDs 0, 1 e 2).

A. Passagem de Token

```
This Server got the Token!
Connecting to Server 10.32.148.54
  Passing Token…

This Server got the Token!
Connecting to Server 10.32.148.54
  Passing Token…

This Server got the Token!
Connecting to Server 10.32.148.54
  Passing Token…
```

[1] Pelo menos e no máximo um servidor deve ter sua constante STARTTOKEN, declarada nos *defines*, definida como 1. Se nenhum servidor possuir esta constante, não haverá Token. E se mais de um possuir, haverá mais de um Token no servidor.

[2] O caminho do arquivo texto deve ser modificado antes da compilação. Para isso, é necessário o acesso ao código-fonte e a modificação na função de chamada de abertura do arquivo, ou seja, *fopen()*.

[3] Após a execução do código do processo *Starter*, é aconselhável a espera do fim da inicialização de todos os servidores antes de inicializar qualquer cliente.

B. Chamada de acesso no Servidor

```
This Server got the Token!
Connecting to Server 10.32.148.54
  Passing Token…
The local Client requested Access to the Critic Session.

This Server got the Token!
Server has the Token. Allowing Client to Write.

Client completed access to file.
Connecting to Server 10.32.148.54
  Passing Token…
```

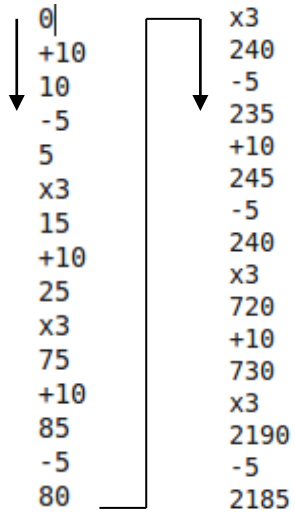
C. Chamada de acesso no Cliente

```
Client requires access to the Critic Session!
Client Authorized. Accessing...
Read from file: 15

Client will add 10 to the previous value.
Writting new value: 25
Client Finished Writting.

Client Finished Access Successfully!
```

D. Arquivo na Sessão Crítica



IV. PERDA DO TOKEN

Em uma parte secundária do projeto, foi definida a possibilidade de implementar uma segurança no sistema para uma possível perda da mensagem Token. Esta perda poderia (certamente) ser causada pela queda de algum servidor. Para isso, algumas mudanças no código apresentado foram implementadas.

A. Algoritmo

Para a detecção da perda da mensagem Token, foi implementado uma idéia baseada no algoritmo de eleição *Bully*, onde todos os servidores se comunicam através de mensagens para todo o sistema (broadcast), perguntando se alguém possui o Token, caso não haja resposta, o servidor de menor *ID* (eleito) cria o seu próprio Token, e segue a execução da estrutura normalmente. O servidor que deixou de responder é descartado.

B. Principais Mudanças

Muitas mudanças foram feitas na impressão dos dados na tela, fazendo a sintaxe do programa ficar mais limpa e de fácil entendimento, porém, as maiores mudanças foram a adição das funções *ConnectThemAll()*, *WhereIsToken()* e *doyouhavetoken()*, auxiliando na verificação da existência do Token.[4]

C. Compilação e execução

A forma de compilação continua praticamente a mesma apresentada anteriormente, a única diferença é que agora, os endereços IP não precisam ser configurados diferentemente em cada servidor, ou seja, o mesmo executável do servidor pode ser utilizado em todas as máquinas, permitindo sua compilação apenas uma vez. Isso foi possível pois, desta forma, como todos os servidores sabem todos os IPs do sistema, não há necessidade de definir ele e o seu próximo separadamente.

Outra mudança foi no cliente *Starter*. Não é mais necessária a declaração dos IPs do sistema antes da compilação, agora, os IPs são adicionados no momento da chamada do programa. Como os servidores são organizados agora por *ID*[5], a ordem da chamada dos IPs pelo *Starter* define a ordem das *ID*s dos processos (0, 1, 2, 3...).

Abaixo é apresentada a sequência de comandos para a execução do sistema:

$ ./server
$ ./starter <Número de servidores> <IP_ID0> <IP_ID1> ...[6]
$ ./client <IPlocal> <ID>

D. Resultados

Este teste foi realizado com três máquinas (*IDs* 0, 1 e 2), onde o servidor de número 0 é derrubado e o sistema passa a comunicar o processo 2 diretamente com o 1, e um novo token é criado pelo 1 (o menor ID). Veja as imagens abaixo.

```
Starting Servers...
Starting server 192.168.1.135 with ID 0
Starting server 192.168.1.136 with ID 1
Starting server 192.168.1.137 with ID 2


  Client Shutting Down!
```

Fig. 3 - Execução do cliente *Starter*

[4] Para melhor esclarecimento de cada procedimento citado, aconselha-se ler o código-fonte do projeto. Neste, há comentários explicando o funcionamento detalhado de cada função.

[5] Como o sistema de *IDs* foi introduzido, não há mais a definição da constante STARTTOKEN, desta maneira, a verificação do servidor que começa com a mensagem é feita pela *ID*, sendo o processo zero o responsável por inicializar o sistema.

[6] A dica para a execução do cliente *Starter* ainda se aplica.

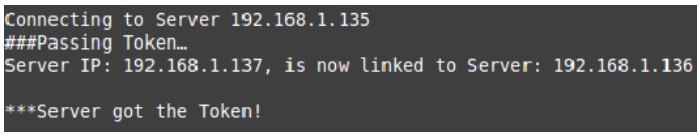


Fig. 4 - Servidor conectado com o próximo servidor online

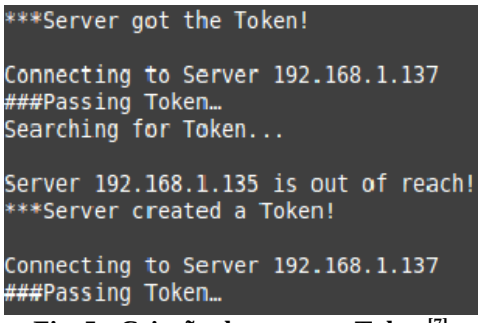


Fig. 5 - Criação de um novo Token[7]

REFERÊNCIAS

[1] Website: Worcester Polytechnic Institude. “Remote Procedure Call (RPC) “. 1995 – 2013. 100 Institute Road, Worcester, MA 01609-2280. http://web.cs.wpi.edu/~cs4514/b98/week8-rpc/week8-rpc.html

[2] Website: Zorzo, Avelino. “Slides das Aulas”. 2013. Moodle PUCRS. http://moodle.pucrs.br/mod/folder/view.php?id=651115

[3] Website: Wikipédia, a enciclopédia livre. “Chamada de procedimento remoto”. 2013. Brasil. http://pt.wikipedia.org/wiki/Chamada_de_procedimento_remoto

[4] Website: Garcia, Luis F. “RPC – Remote Procedure Call”. 2010. http://penta2.ufrgs.br/rc952/trab1/rpc.html

[7] A verificação da existência do Token é feita após 30 segundos sem o recebimento da mensagem.