Servidor para múltiplos clientes

Manassés Ferreira Neto[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais
31270-901 – Belo Horizonte – MG – Brazil

mfer@dcc.ufmg.br

Abstract. .

Resumo. *Do ponto de vista funcional, o objetivo do trabalho é implementar um sistema de envio de mensagens de um servidor que atenda a múltiplas requisições. Do ponto de vista didático, o objetivo é o domínio da programação orientada a eventos utilizando-se a primitiva select e a temporização por sinais.*

1. O problema

A conjectura de Beal é uma conjectura em teoria dos números que generaliza o último teorema de Fermat. Beal formulou a seguinte conjectura:

Se A^x +B^y = C^z, em que A, B, C, x, y, e z são inteiros positivos, com x, y, z > 2, então A, B, e C possuem um fator primo comum.

Por exemplo: 3^3 + 6^3 = 3^5. Nesse exemplo as bases tem fator primo comum 3.

Neste trabalho iremos criar um sistema distribuído para encontrar um contra-exemplo para a conjectura, desenvolvendo um servidor e um cliente simples com troca de mensagens em C++, utilizando comunicação via protocolo UDP.

2. O protocolo

Um cliente se registrará no servidor enviando uma mensagem contendo um identificador do cliente.

O servidor irá confirmar o recebimento da solicitação de registro respondendo com quatro valores inteiros bmin, bmax, pmin, pmax, de agora em diante, chamados de **set**.

Em que bmin e bmax representam os valores mínimo e máximo para as bases (A, B, C) e pmin e pmax representam os valores mínimos e máximos para os expoentes (x, y, e z).

O cliente irá responder o servidor confirmando os valores recebidos.

O cliente deverá procurar por um contra-exemplo da conjectura no intervalo dado testando todos os valores possíveis.

Ao final da procura, o cliente irá notificar o servidor se encontrou ou não um contra-exemplo no intervalo dado, e se encontrou, quais os valores de A, B, C, z, y, z.

Caso algum mensagem seja perdida, é necessário fazer a retransmissão.

Vários clientes serão executados simultaneamente. O servidor precisa atender a todos esses clientes simultaneamente. Para cada cliente que conectar ao servidor, o servidor irá responder com um valor intervalo diferente de bases ou expoentes (bmin, bmax, pmin, pmax). O servidor deve armazenar quais intervalos já foram procurados.

Um navegador será utilizado para monitoramento do servidor. O servidor responderá, utilizando um outro porto, uma página HTML que mostra quais os intervalos já foram procurados e contra-exemplos encontrados .

3. Implementação

Para desenvolver o trabalho foi adotada a linguagem de programação C++, o IDE rápido e leve: geany, e o sistema operacional crunchbang, uma distro linux oriunda do debian.

Para entender a troca de mensagens realizou-se o seguinte esquema:

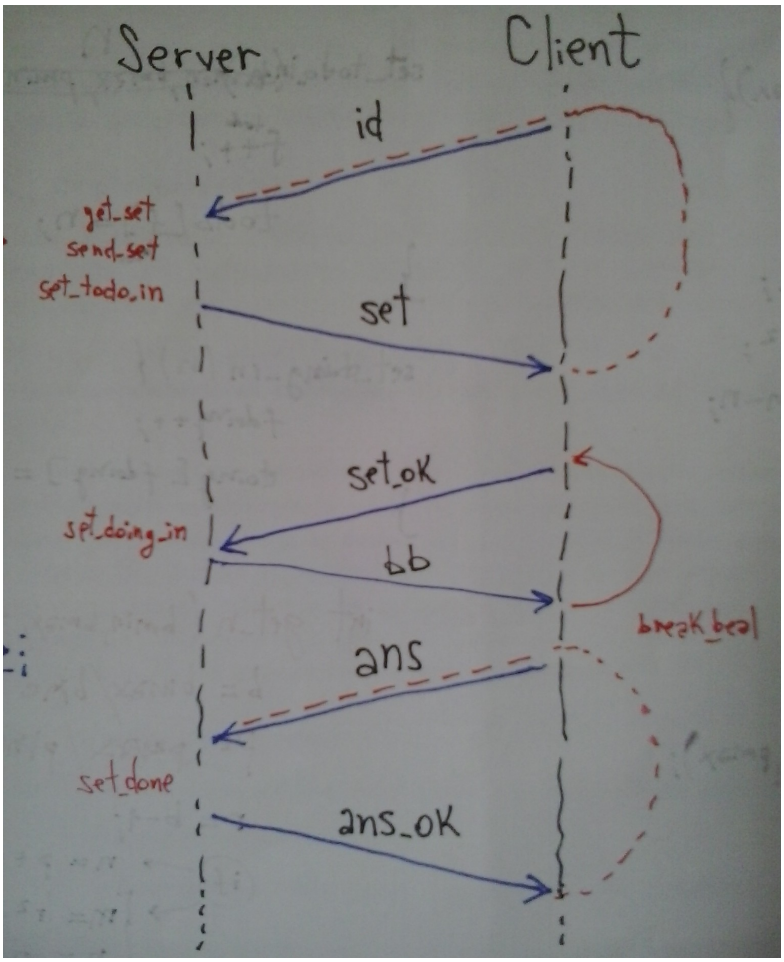

Figura 1. Fluxo das Mensagens

3.1. Decisões de Projeto

Dado o protocolo decidiu-se que:

1. a identificação do cliente ficou abstraída no socket.
2. servidor não retransmite.
3. cliente envia mensagem e aguarda confirmação, se não obtiver, retransmite.
4. a capacidade o servidor responder simultaneamente a múltiplas conexões será obtida com uso de select.
5. a temporização do cliente para retransmitir será implementada por select.

6. o espaço de busca bidimensional (base X expoente) será percorrido subdividindo-o em dois 'triângulos', e alternando entre esses.
7. O controle do que já foi feito, do que está em andamento e do que começou a ser feito e não terminou será feito em:
    1. uma variável *sid* (que determina o número do último set a ser solicitado)
    2. duas filas: *todo* e *doing.*
    3. um *conjunt*o com os valores dos sets de *todo* e de *doing*.
8. Quando o servidor fica fora do ar, os clientes, quando em localhost, interrompem-se na primeira tentativa de comunicação. No entanto, remotamente realizam *maisuma (10)* tentativas para a etapa atual e buscam identificar-se novamente por *maisuma (5)* vezes.
9. A mensagem sai do cliente comunica-se por portas altas aleatórias e chega numa bem definida no servidor.
10. A troca de mensagem seria feita com um comando inicial, orientando a etapa envolvida, além disso, teria também o *sid* para permitir identificar o intervalo.

3.2. Estruturas de dados

No servidor estão as estruturas mais interessantes:

```
std::set<BigInteger> todo;

std::set<BigInteger> doing;

std::set< std::vector<BigInteger> > conjunto;
```

Todas as três fazem uso da biblioteca de grandes inteiros de Matt Cutchen. São resultado direto das decisões tomadas e descritas na seção anterior.

3.3. Algoritmos principais

O código fonte do servidor e dos clientes possuem comentários que identificam as partes do algoritmo de alto nível e explicam os detalhes das partes principais do programa. Do lado do servidor, a chamada select permitiu a manutenção de múltiplos clientes.

Vale destacar que, no código do cliente, para testar a conjectura, para o dado intervalo, testou-se por força bruta todas as combinações e usou-se o máximo divisor comum para determinar se Beal havia sido quebrado. Isto é se A, B e C possuiam mdc(A,B,C) igual a um. Uma vez que 'possuir fator primo comum é o mesmo que possuir mdc maior que um'.

3.4. Compilação e Execução

Junto do código fonte, há um arquivo Makefile, para auxiliar na compilação, e um arquivo README, que descreve como usar os programas client e server.

4. Testes

A seguir, imagens da tela mostrando o correto funcionamento com múltiplos clientes e o retorno da página HTML de gerência do servidor.

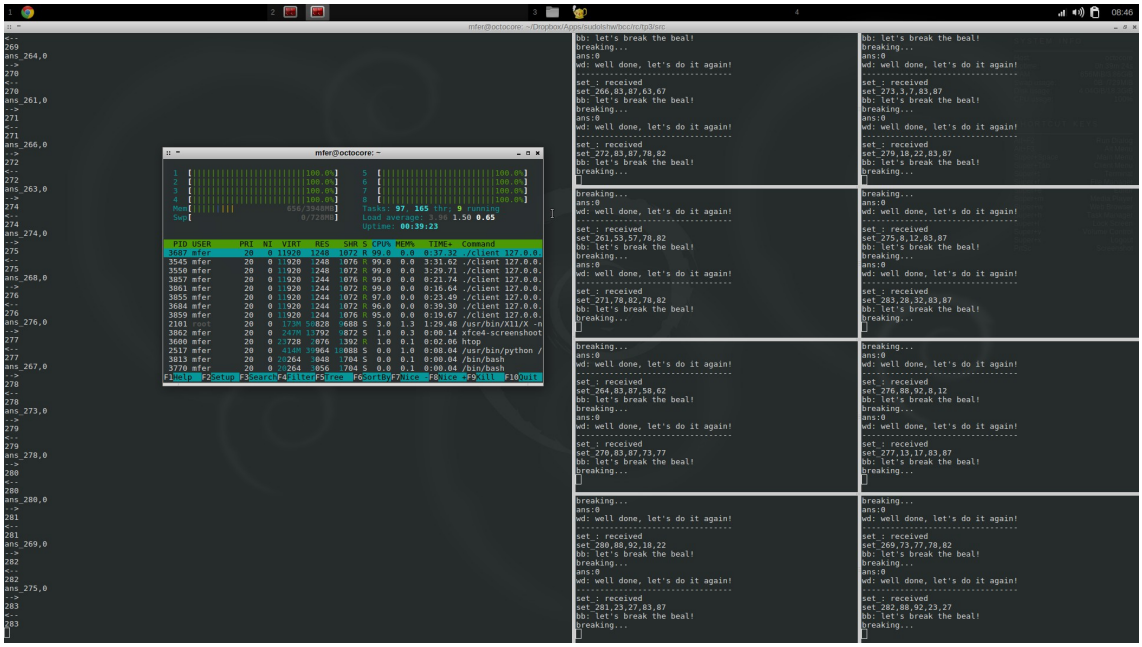

Figura 2. Ilustrando atividade de oito clientes

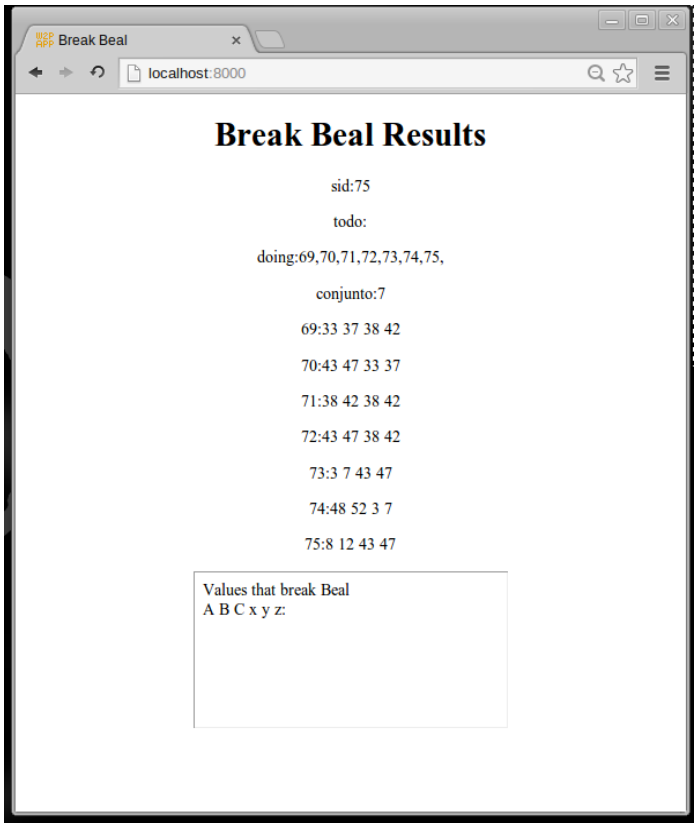

Figura 3. Página HTML para monitoramento.

5. Conclusões

Esse trabalho concretizou-se uma experiência muito rica no que diz respeito as comunicação em rede e distribuição de processamento.

Entre as modificações futuras, que não puderam ser feitas até a entrega do trabalho, estão:

- a gerência dos sets em *doing* que após um dado tempo não foram feitos, devendo ser movidos para a fila *todo*. Ao invés da heurística usando CLIENTS.
- a leitura do arquivo de persistência, no início da execução do servidor, para que os cálculos não sejam repetidos.
- Coloar no servidor o módulo para conferir um dado ersultado antes de anunciá-lo.

6. Referências

O código base para o servidor/cliente UDP foi retirado do manual do getaddrinfo. O exemplo de uso do select e da temporização foram retirados da página pessoal de André Moreira.

O manual do select foi bastante útil para entender como usar a multiplexação síncrona, principalmente, por conta do exemplo.

CUTCHEN, Matt, C++ Big Integer Library, https://mattmccutchen.net/bigint/bigint-2010.04.30.zip

Linux man page, select(2), http://linux.die.net/man/2/select, Acesso em Dezembro 2013.

Linux man page, getaddrinfo(3), http://linux.die.net/man/3/getaddrinfo, Acesso em Dezembro 2013.

Moreira, André – Instituto Politécnico do Porto - Página pessoal, http://www.dei.isep.ipp.pt/~andre/documentos/sockets-berkeley.html, http://www.dei.isep.ipp.pt/~andre/documentos/samples/udp_cli3.c, Acesso em Dezembro 2013