CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

***Curso**: Engenharia de Computação*
***Disciplina**: Sistemas Distribuídos*
***Ano/Semestre**: 2017/2*
***Professora**: Anolan Yamilé Milanés Barrientos*
***Atividade**: Trabalho Prático 1 - Implementação de um serviço de consultas a um log distribuído*

Arley dos Santos Ribeiro
Bruno Marques Maciel
Cassiano de Brito Andrade

1. PROJETO DO PROGRAMA

Desenvolveu-se o código em C++ e optou-se pela abordagem *multithreading*, ou seja, ao chegar uma nova requisição no servidor, este cria uma nova *thread* para atendê-la e, após seu término, esta *thread* é finalizada. Observa-se, nesta abordagem, que há um bom aproveitamento dos recursos computacionais, por haver compartilhamento de recursos entre as *threads*, ao passo que os clientes são atendidos sob demanda, garantindo um certo grau de escalabilidade. Como as *threads* realizam apenas leitura, não há risco de resultados inesperados (leitura é idempotente) e nem há concorrência entre as mesmas.

Em comparação com a abordagem de processamento único, a abordagem *multithreading* tem a vantagem de poder atender múltiplos clientes simultaneamente. Entretanto, se não for definida uma quantidade máxima de *threads* que podem estar em execução na abordagem *multithreading*, pode-se sobrecarregar o servidor se este for exposto a uma elevada quantidade de requisições, a ponto de causar *crash* no mesmo. Isto não aconteceria na abordagem de processamento único, pois nesta atende-se uma requisição por vez, porém não tem-se nenhuma escalabilidade.

A abordagem multiprocessing, por sua vez, possui maior confiabilidade por não conter um ponto único de falha (PUF), porém consome muita memória, pois cada processo é independente e possui sua alocação de memória própria.

2. FORMATO DAS MENSAGENS

A mensagem enviada do cliente para o servidor possui, somente, o comando a ser executado e a quantidade de caracteres que este possui. A resposta do servidor ao cliente possui o resultado deste comando ao ser executado no próprio servidor, assim como o tamanho em caracteres do resultado.

Tabela 1 - Composição das mensagens trocadas entre as entidades

Mensagem enviada do cliente ao servidor	comando *grep*
Mensagem enviada do servidor ao cliente	resposta do comando *grep*

Criou-se um arquivo texto com o endereço IP e porta de todos os servidores separados por linha, de forma que, quando o cliente vai executar, ele abre este arquivo e armazena em uma variável local os endereços IPs e portas dos servidores.

Tabela 2 - Exemplo de arquivo que contém lista de todos os servidores

Endereço IP	Porta
192.168.1.114	8880
192.168.1.114	8881
192.168.1.114	8882
192.168.1.114	8883

3. *UNIT TESTS*

3.1 Teste de um cliente e um servidor em uma mesma máquina virtual

Inicialmente, testou-se a funcionalidade dos códigos em uma mesma máquina virtual, com o servidor sendo executado em um processo e o cliente em outro (terminais diferentes).

Este caso de teste visa garantir que:
1. O servidor irá aguardar por requisição, eventualmente atender uma requisição que chegar e retornar ao estado de escuta;
2. O cliente irá iniciar uma conexão com o servidor passando o comando desejado;
3. O servidor irá conseguir executar o comando e retornar a resposta ao cliente.

3.2 Teste de um cliente e múltiplos servidores em uma mesma máquina virtual

Executou-se em uma única máquina virtual quatro servidores e um cliente. Portanto, este caso de teste tem o intuito de testar a comunicação do cliente com múltiplos servidores, ou seja, a administração das requisições assim como a administração das respostas.

3.3 Teste de um cliente e múltiplos servidores em máquinas virtuais distintas

Para avaliar o sistema distribuído, planejou-se executar tanto o cliente quantos os servidores em máquinas virtuais distintas, de forma que o endereço IP de cada uma delas fosse diferente. Assim, procura-se testar a comunicação entre os processos pela rede local.

3.4 Teste de um cliente e múltiplos servidores em máquinas virtuais distintas consultando arquivo log de 100MB

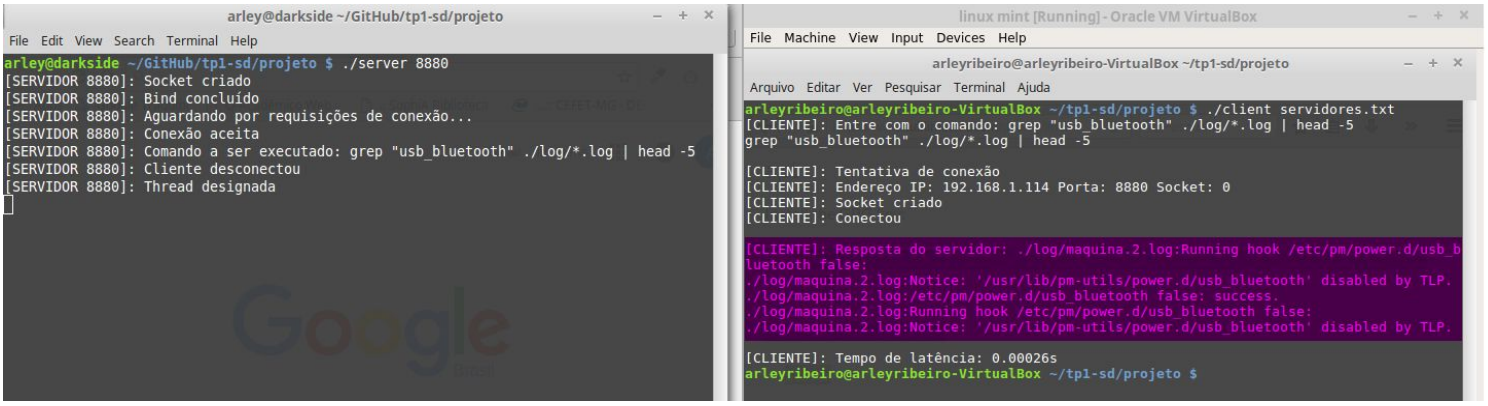
Neste caso de teste, pretende-se avaliar o tempo de latência da comunicação, desde o momento de requisição do cliente até a obtenção da resposta, em função de arquivo de log de 100MB por servidor.

4. RESULTADOS ENCONTRADOS

4.1 Teste de um cliente e um servidor em uma mesma máquina virtual

Pela Figura 1, observa-se que o cliente requisitou o comando `grep “usb_bluetooth” ./log/*.log | head -5`, ou seja, buscou-se por *matches* com o texto “usb_bluetooth” nos arquivos de extensão *log*, solicitando os cinco primeiros resultados. A resposta obtida pode ser observada no destaque em rosa na figura. A requisição teve uma latência de $260\mu s$.

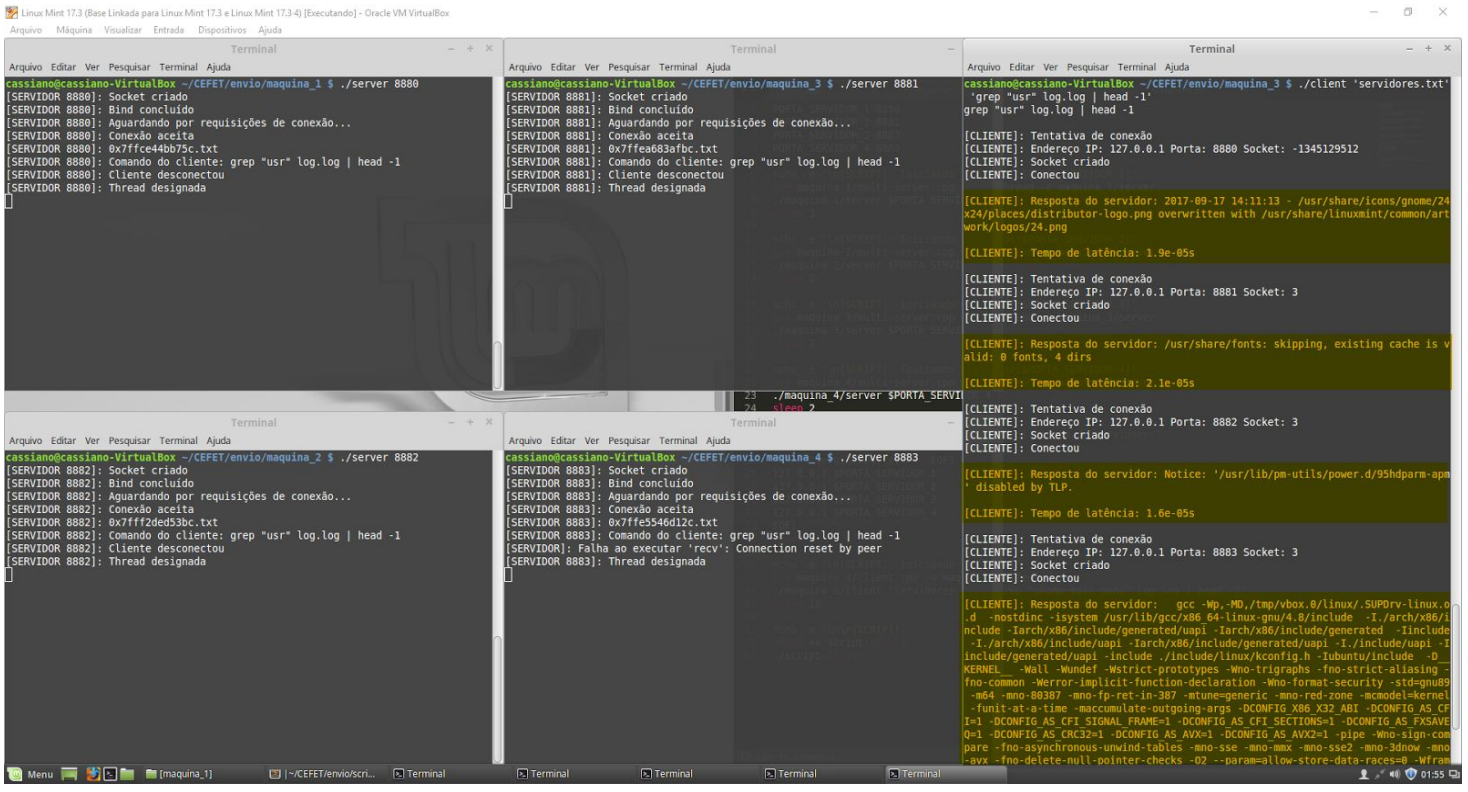
Figura 1 - Resultado obtido para o caso de teste especificado na Seção 3.1



4.2 Teste de um cliente e múltiplos servidores em uma mesma máquina virtual

Rodou-se quatro servidores, um em cada terminal, e o cliente em um quinto terminal. Solicitou-se pelo comando `grep “usr” log.log | head -1`, e obteve-se as quatro respostas, como destacado em amarelo.

Figura 2 - Resultado obtido para o caso de teste especificado na Seção 3.2



4.3 Teste de um cliente e múltiplos servidores em máquinas virtuais distintas

Executou-se os servidores em três máquinas virtuais distintas, ao passo que o cliente foi executado na máquina real. Com isso, observa-se na Figura 3 que o cliente (1º terminal) tem impresso a resposta do primeiro servidor (3º terminal) e do segundo servidor (4º terminal), ao passo que na Figura 4 já é possível ver a resposta do terceiro servidor (2º terminal).

Figura 3 - Primeira parte do resultado obtido para o caso de teste especificado na Seção 3.3

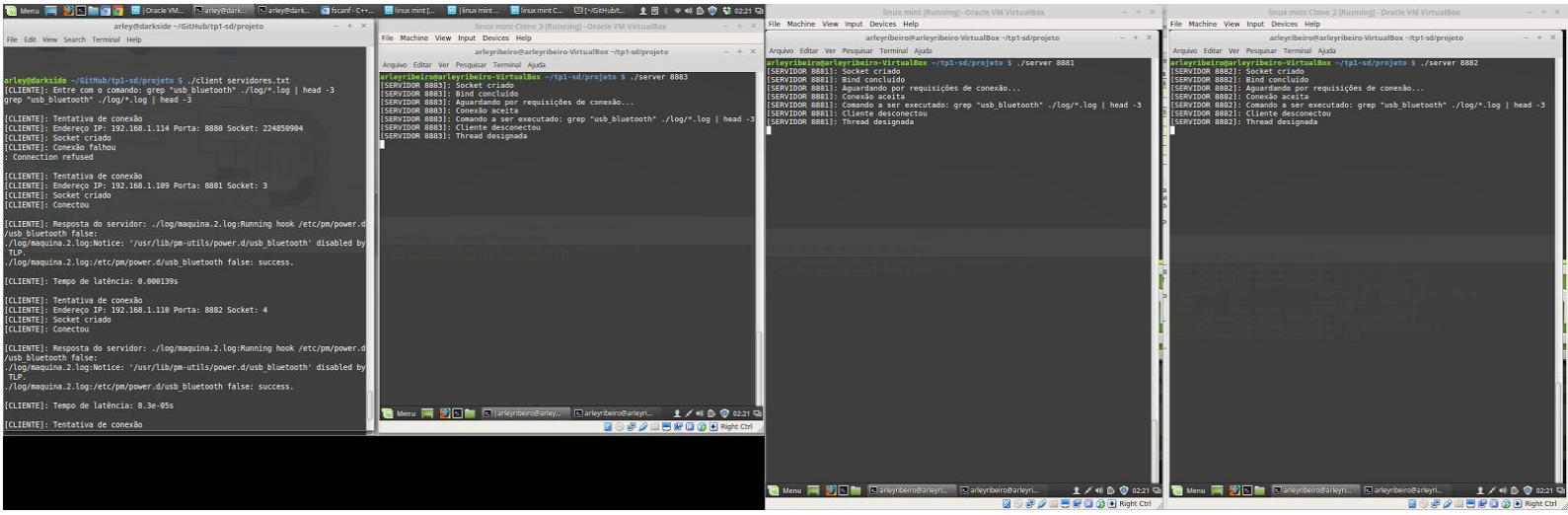
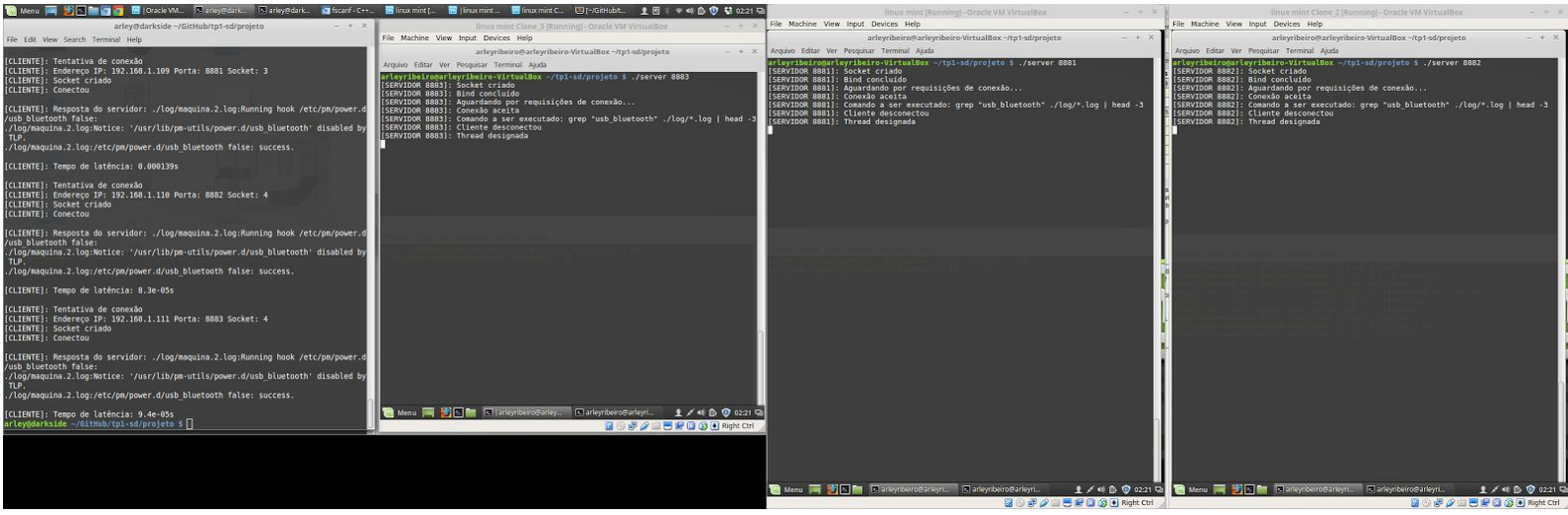


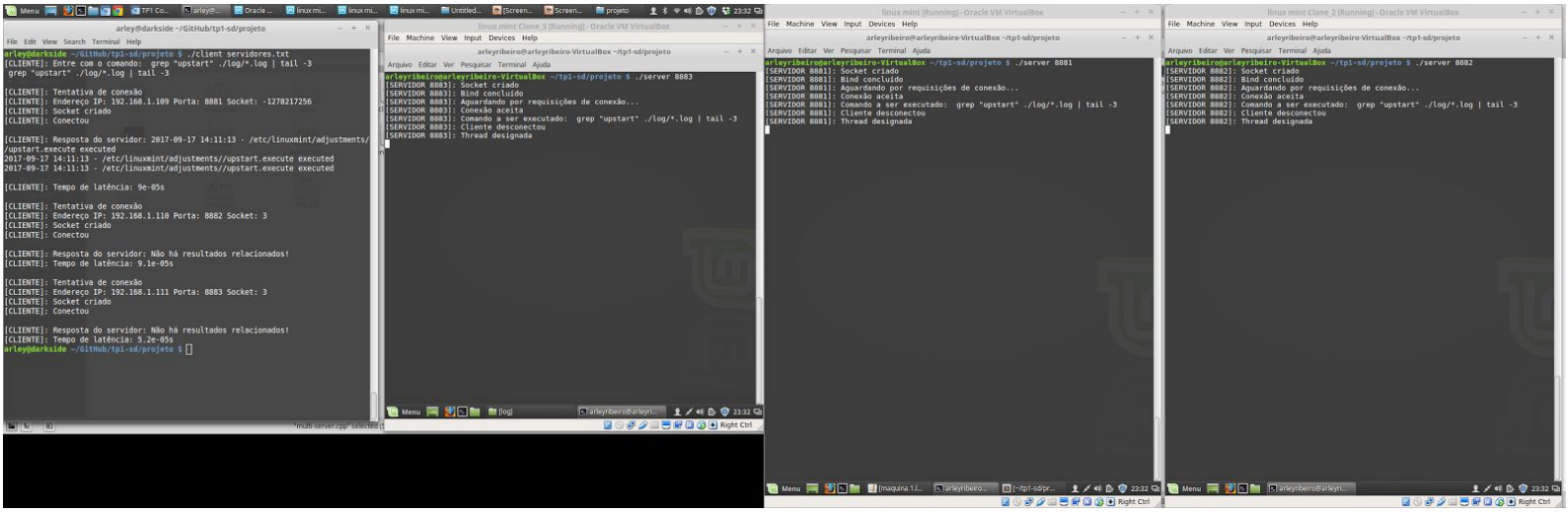
Figura 4 - Segunda parte do resultado obtido para o caso de teste especificado na Seção 3.3



4.4 Teste de um cliente e múltiplos servidores em máquinas virtuais distintas consultando arquivo log de 100MB

Executou-se os três servidores e o cliente em máquinas virtuais distintas, de forma a analisar o tempo de latência quando cada servidor consulta arquivos de 100MB cada. O resultado obtido pode ser visualizado na Figura 5, de forma que o primeiro terminal é referente ao cliente e os demais ao servidor.

Figura 5 - Resultado obtido para o caso de teste especificado na Seção 3.4



5. AVALIAÇÃO DE DESEMPENHO E CONCLUSÕES

Para o sistema implementado, observa-se que a complexidade associada ao tempo de requisição do cliente é $O(s)$, sendo s a quantidade de servidores, devido à iteração linear sobre a lista dos mesmos. Este método utilizado não é o mais eficiente, pois para um s muito grande o tempo de requisição será muito alto. No entanto, ele foi adotado devido à facilidade de implementação e depuração.

O tempo de atendimento do servidor possui complexidade $O(f)$, sendo f o tamanho total dos arquivos que ele terá de analisar. Observa-se que o número de clientes não entra na função de complexidade pois o servidor aplica a abordagem *multithreading*, ou seja, dado um número de clientes ideal, nenhum deles deverá enfrentar fila.

Na Tabela 3, observou-se entre os casos de teste 3.2 e 3.4 que o tempo de latência aumentou consideravelmente no caso 3.4, pois este abordou a situação em que os arquivos de log possuíam em torno de 100MB cada. O caso de teste 3.1 teve um tempo de latência alto inexplicado sendo possível justificá-lo como sendo um *outlier* fruto de algum evento inesperado de processamento.

Tabela 3 - Exibição do tempo de latência médio para cada caso de teste

Caso de teste	Tempo de latência médio (10^{-3} segundos)
3.1	0,260
3.2	0,019
3.3	0,105
3.4	0,233

Como o sistema foi executado em uma rede local, observou-se que o tempo de latência possui seu gargalo no processamento dos servidores, pois o tempo de transmissão é baixíssimo. Com isso, destaca-se que uma possível otimização para o sistema implementado é tornar o cliente *multithreading*, pois desta forma este poderia enviar requisições paralelas aos servidores, deixando cada *thread* administrar a sua respectiva conexão para, no final, unir todas as respostas oriundas de cada *thread* em uma só.

REFERÊNCIAS

ADVANTAGES and Disadvantages of a Multithreaded/Multicontexted Application. Disponível em: <https://docs.oracle.com/cd/E13203_01/tuxedo/tux71/html/pgthr5.htm>. Acesso em: 17 set. 2017.

C++ multi threaded server and client. 2013. Disponível em: <http://www.cplusplus.com/forum/unices/116977/>. Acesso em: 17 set. 2017.

SERVER and client example with C sockets on Linux. 2012. Disponível em: <http://www.binarytides.com/server-client-example-c-sockets-linux/>. Acesso em: 17 set. 2017.

SOCKET Programming in C/C++. Disponível em: <http://www.geeksforgeeks.org/socket-programming-cc/>. Acesso em: 17 set. 2017.

SOCKETS Tutorial. Disponível em: <http://www.linuxhowtos.org/C_C++/socket.htm>. Acesso em: 17 set. 2017.