Redes de Computadores e Sistemas Distribuídos: EP1

Apresentação do segundo Exercício Programa da disciplina

Aluno: Luis Vitor Zerkowski
NUSP: 9837201
Professor: Daniel Batista
Disciplina: MAC0352

Objetivo

- O objetivo do programa é construir um servidor - também conhecido como broker - para clientes do protocolo MQTT - versão 5.

- Para cumprir com o propósito da tarefa, utilizei o servidor fornecido pelo próprio professor fazendo apenas os devidos ajustes no segmento de código indicado no arquivo.

Arquitetura e Implementação

- A principal decisão de arquitetura a ser tomada tratava da forma de intercomunicação entre os *sockets*.

- Para cumprir tal missão, resolvi criar arquivos referentes aos tópico. Dessa forma, um novo arquivo é criado sempre que um novo tópico surge e, caso um tópico já existente seja referenciado, o seu respectivo arquivo será aberto em modo de leitura.

Arquitetura e Implementação: Inscrição

- Os processos do servidor que servem aos clientes de inscrição, portanto, dividem-se entre verificar se o cliente enviou uma requisição de desconexão e verificar se o arquivo referente ao tópico do cliente inscrito foi atualizado.

- Em caso de requisição de desconexão, o programa sai do laço e o processo referente ao cliente termina. Já no caso de alteração de arquivo - causada por uma publicação no devido tópico -, a última linha do arquivo é lida e escrita nos clientes inscritos no respectivo tópico.

Arquitetura e Implementação: Inscrição

- Principais vantagens da arquitetura:
  - Rápida busca pelo tópico de inscrição - não precisa seguir um protocolo de busca dentro de um único arquivo para achar o tópico.
  - Rápida busca pela nova publicação - não precisa seguir um outro protocolo de busca dentro de um único arquivo após já ter achado o tópico, basta ler a última linha até o *'\n'*, caractere que indica o final da publicação.
  - Escalável e paralelizável - permite que vários processos leiam o mesmo arquivo ao mesmo tempo.
  - Implementação bastante simples.

Arquitetura e Implementação: Inscrição

- Principais desvantagens da arquitetura:
  - Não permite hierarquia de tópicos.
  - Apesar de escalável e paralelizável, apresenta suas limitações devido ao tempo necessário antes do *timeout* da função *read* - se múltiplos clientes de publicação publicam num mesmo tópico com diferença de tempo menor do que o tempo do *timeout*, 0.001 segundo no caso de meu programa, apenas a última mensagem é lida pelos clientes inscritos naquele tópico.

Arquitetura e Implementação: Publicação

- Os processos do servidor que servem aos clientes de publicação, por sua vez, são um pouco mais simples, apenas escrevendo suas mensagens no arquivo referente ao tópico escolhido com um *'\n'* para indicar finalização da publicação.

- Após escreverem sua mensagem no arquivo em questão, esses processos são automaticamente finalizados, fechando, pois, a conexão com o cliente de publicação.

- Vale destacar, ainda, que para evitar perda de mensagens, foi feito o uso da função *'fcntl'* que funciona como um semáforo para arquivos.

Arquitetura e Implementação: Publicação

- Principais vantagens da arquitetura (análogas às vantagens da arquitetura de inscrição):
  - Rápida busca pelo tópico da mensagem - não precisa seguir um protocolo de busca dentro de um único arquivo para achar o tópico.
  - Rápida escrita - não precisa seguir um outro protocolo de busca dentro de um único arquivo após já ter achado o tópico, basta abrir o arquivo em modo *append* e escrever a mensagem com um '\n' no final.
  - Escalável e paralelizável - permite que vários processos escrevam no mesmo arquivo ao mesmo tempo.
  - Implementação bastante simples.

Arquitetura e Implementação: Publicação

- Principais desvantagens da arquitetura (análogas às desvantagens da arquitetura de inscrição):
  - Não permite hierarquia de tópicos.
  - Apesar de escalável e paralelizável, apresenta suas limitações devido ao tempo necessário antes do *timeout* da função *read* - se múltiplos clientes de publicação publicam num mesmo tópico com diferença de tempo menor do que o tempo do *timeout*, 0.1 segundo no caso de meu programa, apenas a última mensagem é lida pelos clientes inscritos naquele tópico.

Experimentos de uso de CPU e Rede

- Para realizar os experimentos de uso de CPU, utilizei o comando *psrecord*, comando esse capaz de registrar o uso de CPU e memória de um processo por um certo intervalo de tempo.

Exemplo de uso do comando:

$psrecord <pid_do_servidor> --plot <nome_do_arquivo>.png --duration <segundos> --include-children

- <pid_do_servidor> indica o identificador do processo do *broker*. <nome_do_arquivo> indica o nome do arquivo no qual se quer salvar o gráfico obtido. <segundos> indica o número de segundos da captura do processo. --include-children indica para o comando que estamos interessados nos processos filho - importante em nossa aplicação.

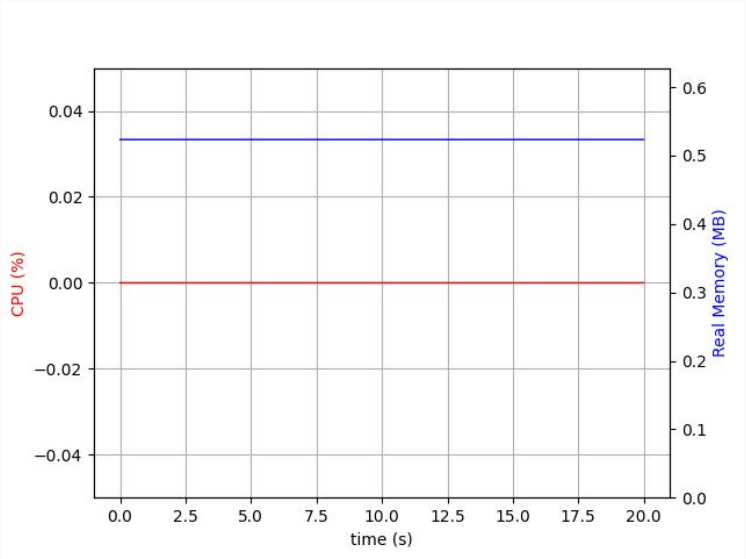
Experimentos de uso de CPU e Rede

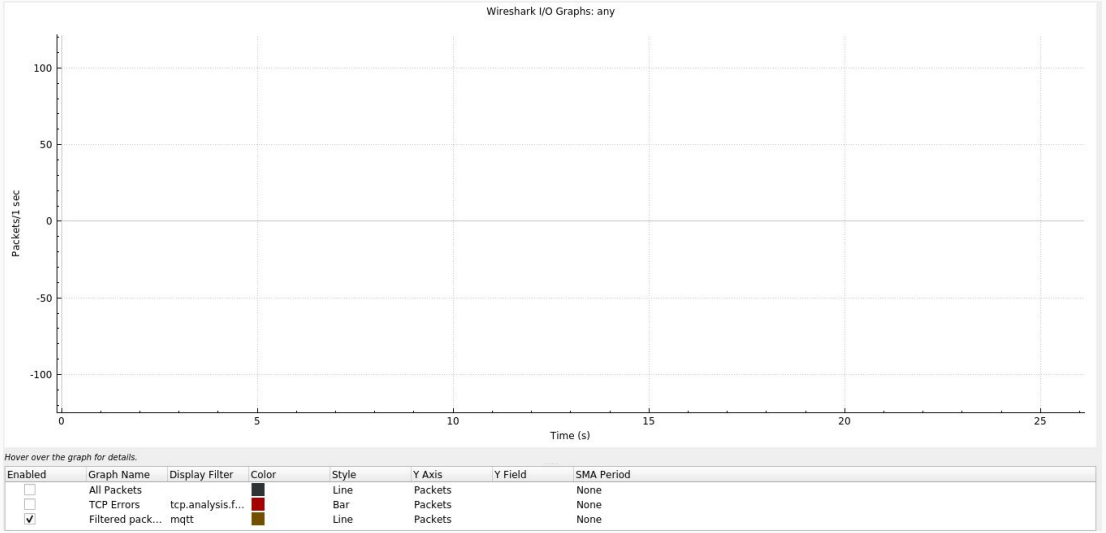
- Para realizar os experimentos de uso de rede, utilizei o próprio Wireshark. Dentro do programa, bastou filtrar os segmentos com o protocolo MQTT e depois gerar o gráfico de rede - entrada e saída - apenas para os pacotes relativos a esse filtro.

- OBS: Todos os experimentos foram feitos com clientes de inscrição e publicação simultâneos, mas após todos os clientes de publicação mandarem suas mensagens para seus respectivos tópicos, um tempo de 0.5 segundos sem nenhuma publicação era acionado para não deixar o número de clientes de publicação crescer exponencialmente.

Experimentos de uso de CPU e Rede

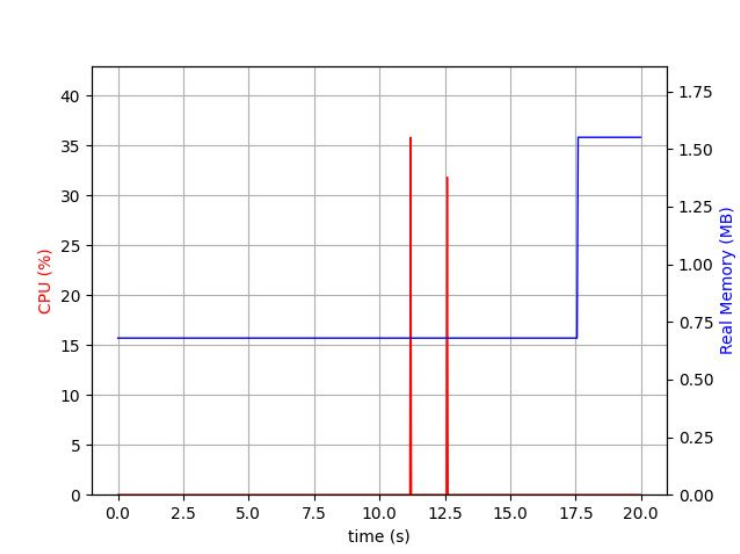
(I) Apenas com o servidor, sem nenhum cliente conectado

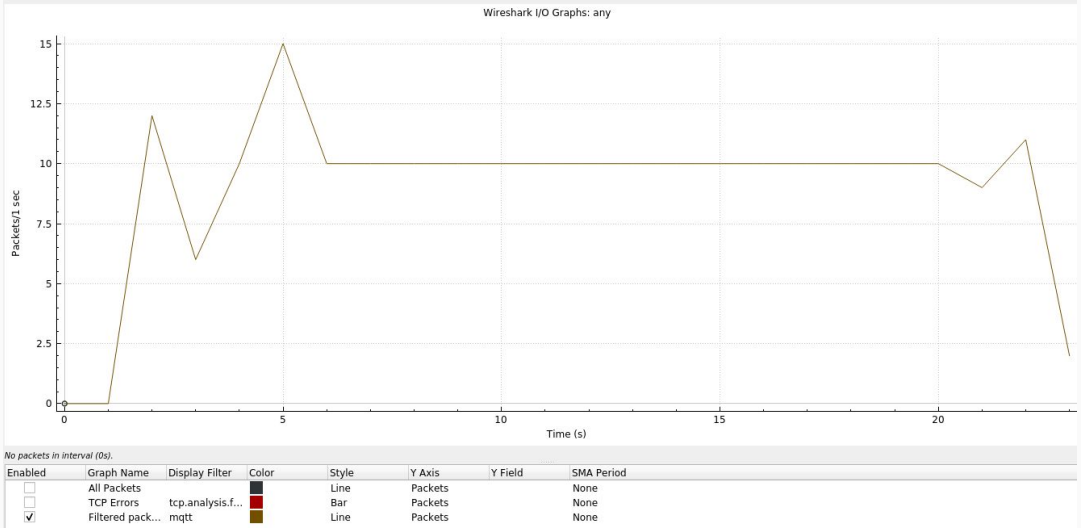




Experimentos de uso de CPU e Rede

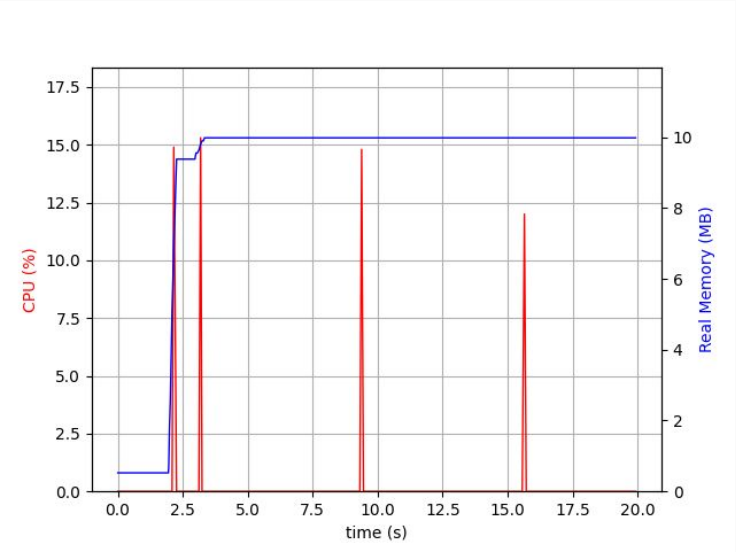
(II) servidor com 2 clientes de inscrição e 2 clientes de publicação

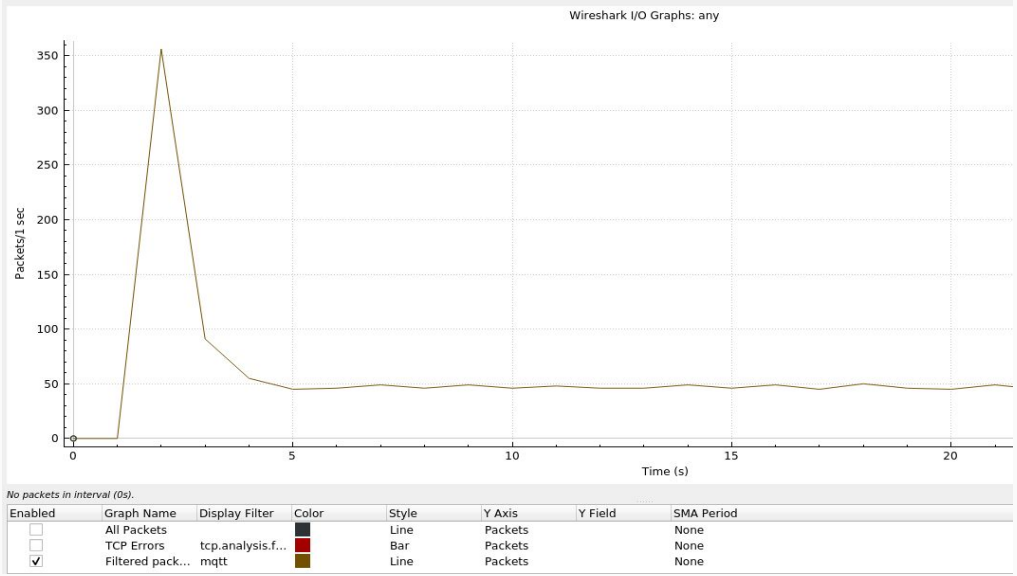




Experimentos de uso de CPU e Rede

(III) servidor com 100 clientes de inscrição e 100 clientes de publicação





Obrigado pela atenção :)