EP1-MAC0352

Implementação de um servidor MQTT

Leandro Rodrigues da Silva
nUSP: 10723944

Introdução

- Para executar o servidor, basta rodar **make**, ou então **gcc ep1_leandro_rodrigues_da_silva.c -o my_mqtt_server && ./my_mqtt_server <porta>**
- O servidor foi implementado em C e testado em uma máquina Linux
- O arquivo **LEIAME** possui mais detalhes de execução

Arquitetura

- A estratégia de comunicação entre os processos escolhida foi a de shared files
- O modelo do servidor é o de **publish/subscribe**
- Um processo pai cria forks para cada requisição recebida (**publish** ou **subscribe)**
- Cada processo filho é responsável por lidar com a tarefa de um cliente

Publishers

- **Publishers** são clientes que enviam uma mensagem para um determinado tópico
- Um cliente criado utilizando o comando **mosquitto_pub -t “topicoX” -m “mensagemY”** irá escrever num arquivo chamado **topicoX** a mensagem **mensagemY**
- Se o arquivo **topicoX** já existe, o **publisher** irá escrever a mensagem no final dele. Caso contrário, o arquivo será criado na pasta onde o programa está rodando

Subscribers

- **subscribers** são clientes que enviam ao servidor um **tópico** que desejam ficar “escutando”
- Sempre que um **publisher** envia uma mensagem para o respectivo tópico, todos **subscribers** daquele tópico recebem essa mensagem.

Subscribers

- Um **subscriber** criado executando **mosquitto_sub -t “topicoX”** ficará lendo um arquivo chamado **topicoX**, aguardando a chegada de mensagens
- Se o arquivo existir, o **subscriber** irá posicionar o ponteiro de leitura no fim, para que antigas mensagens não sejam entregues a ele. Caso contrário, o arquivo será criado

Desempenho

- Foi analisado o desempenho do servidor em três situações:

  1. Executando sem requisições de clientes

  2. Executando com um cliente em subscribe e outro cliente publicando uma mensagem

  3. Executando com 50 clientes em subscribe e 50 clientes publicando mensagens

Método de análise

Para analisar o desempenho em cada cenaŕio, foi utilizado o seguinte método:

1. Eliminação do máximo de processos rodando simultâneamente para evitar interferencias

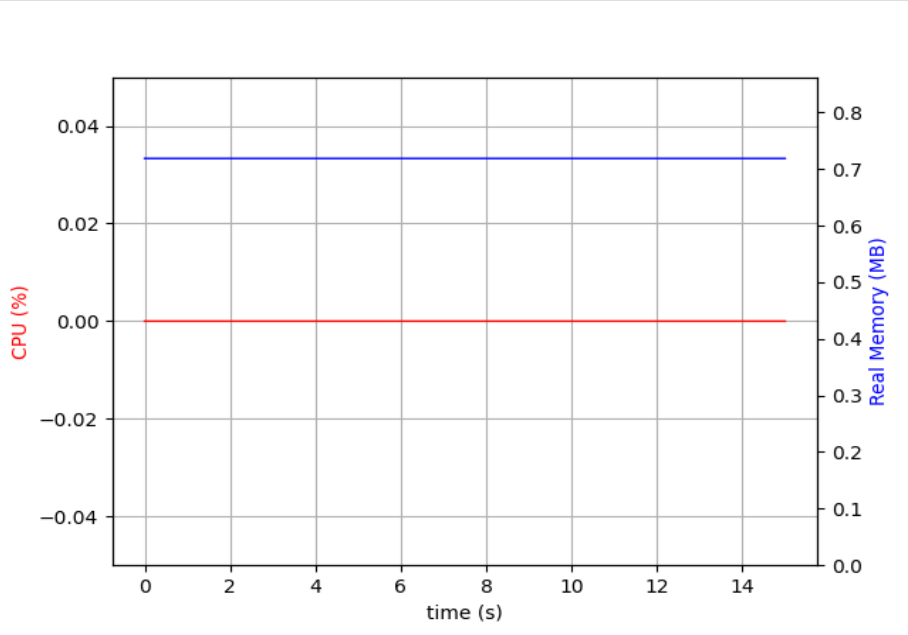
2. Execução do servidor junto a execução do programa **psrecord**, responsável por plotar um gráfico do uso de CPU de um dado programa

3. Captura dos pacotes utilizando o programa **tshark** para gerar um arquivo .**pcapng** para ser aberto no **Wireshark** e gerar os gráficos de tráfego. A captura foi feita utilizando a interface de loopback com o filtro da porta utilizada pelo servidor para evitar interferências

4. Execução de um shell script que executa uma quantidade X de clientes

Resultados

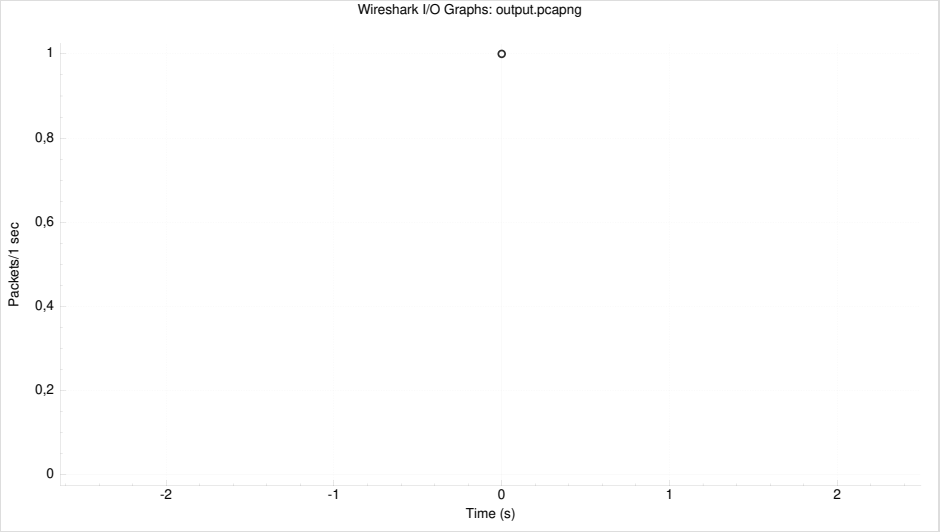
1. Executando o servidor sem clientes



- Sem clientes, o servidor manteve o uso de CPU em 0%. Como o processo pai fica em um loop infinito aguardando pela conexão dos clientes, sem executar nenhuma ação, o sistema operacional não escalona ele e o uso de CPU não aumenta.

Resultados

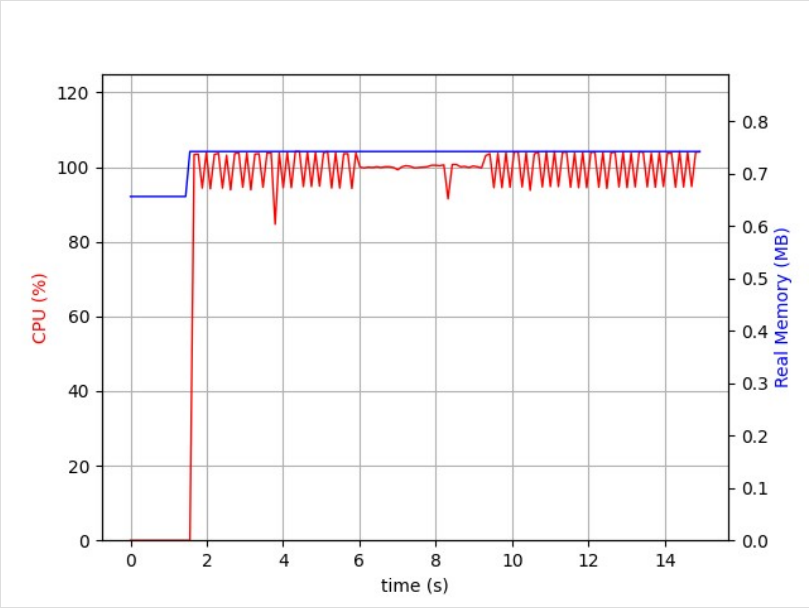
1. Executando o servidor sem clientes



Como o cliente não recebe nenhum pacote nesse cenário, o wireshark não captura nenhum tráfego

Resultados

2. Executando o servidor com dois clientes



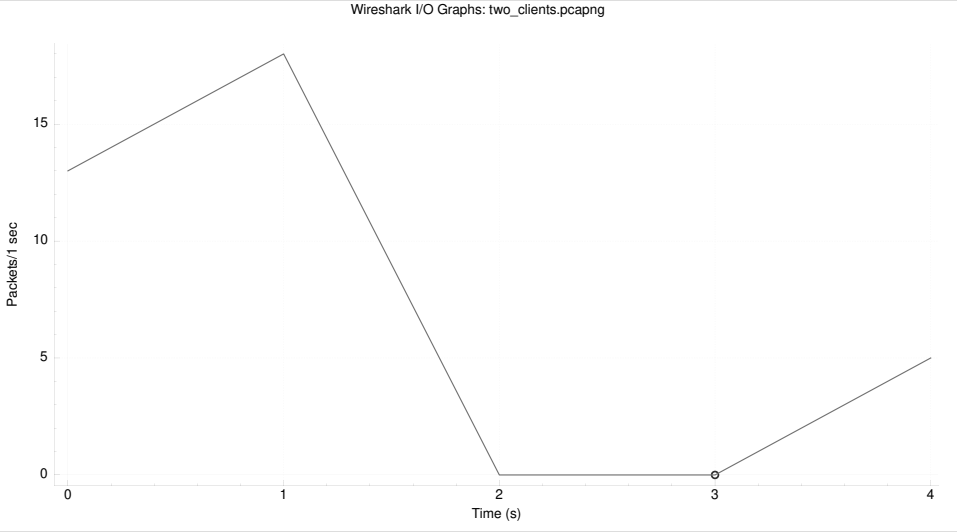
- Com dois subscribers rodando simultaneamente e o envio sequencial de mensagens por publishers, o uso de processamento foi consideravelmente alto

- O uso no início da execução começa em 0%, pois o processo pai está em espera, e com a chegada das requisições dos clientes, aumenta abruptamente

- O uso passa de 100% pois o programa de plot foi executado com a flag –**include-children**, que resulta na soma o processamento do servidor com o dos seus processos filhos. Sem isso, seria necessário capturar o processamento de cada subprocesso individualmente

Resultados

2. Executando o servidor com dois clientes



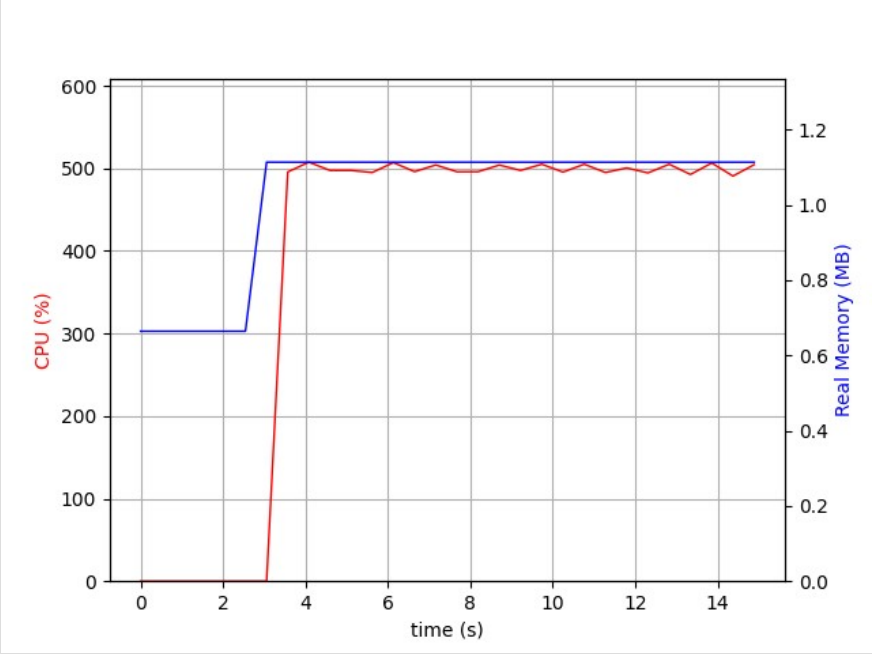
- O tráfego utilizando um subscriber e um publisher começa alto devido a chega das requisições dos dois clientes

- Logo em seguida, sem nenhum cliente novo chegando, o tráfego vai para zero

- Por fim, com a interrupção do servidor, o pacote de disconnect é enviado ao subscriber conectado, gerando um pouco mais de tráfego

Resultados

3. Executando o servidor com 100 clientes

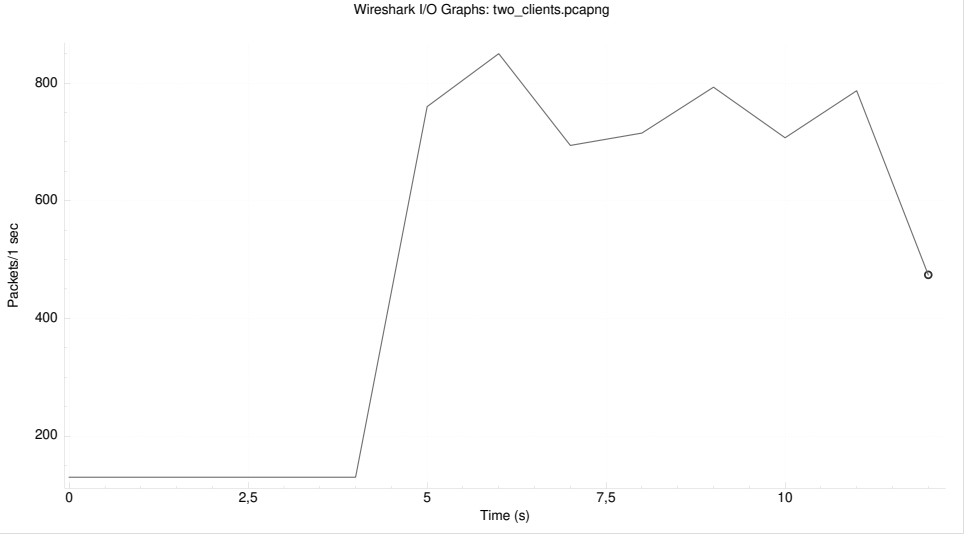


- Utilizando 50 publishers e 50 subscribers, o consumo de CPU chegou em torno de 500%

- Foram inicializados primeiro os subscribers, e pois foram disparadas 50 mensagens para eles, todos no mesmo tópico com a mesma mensagem

Resultados

3. Executando o servidor com 100 clientes



- Utilizando 50 publishers e 50 subscribers, cerca de 6000 pacotes dos clientes e do servidor trafegaram na rede

- O gráfico começa com tráfego 0, até começar a receber os primeiros clientes.

Após 10 segundos, os publishers param de chegar e o processo é encerrado, enviando os últimos 50 pacotes de disconnect (um para cada subscriber)

Pós-execução

Quando um CTRL+C é pressionado no shell executando o servidor, ele executa os seguintes passos :

- Deleta os arquivos criados para cada tópico
- Libera a memória necessária
- Envia os pacotes disconnect para cada subscriber remanescente

Limitações

O servidor possui algumas limitações:

- Os tópicos e mensagens devem possuir até 128 caracteres cada, pois apenas o byte menos significativo do length destes está sendo considerado.
- O servidor foi testado apenas com caracteres ASCII nas mensagens e nos tópicos
- O servidor assume que nenhum outro processo irá escrever nos arquivos de tópicos enquanto ele executa, então não lida com condições de corrida entre múltiplos servidores alterando um mesmo tópico.

Fontes e recursos

- ***How to monitor CPU/memory usage of a single process?*** *https://unix.stackexchange.com/questions/554/how-to-monitor-cpu-memory-usage-of-a-single-process*
- ***Psrecord -*** *https://github.com/astrofrog/psrecord*
- ***MQTT V5 RFC -*** *https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html#_Toc3901100*