EP1 de MAC0352

Implementando um Broker MQTT

por Fernando Henrique Junqueira Muniz Barbi Silva
n° USP 11795888

Observações sobre a apresentação

Apresentarei o trabalho em alto nível, ou seja, omitindo detalhes de implementação no código, e falando de forma mais intuitiva, procurando facilitar o entendimento humano.

O código do projeto está bem organizado, e, portanto, não é necessário abordar instruções da linguagem quando não forem decisões de projeto importantes.

Esta será uma breve apresentação ;)

Considerações Iniciais

- Como base para esse projeto, foi utilizado o código com pipes disponibilizado pelo professor no e-Disciplinas.
- Ou seja, também adotei a estratégia de forks.
- O broker realiza o envio de mensagens entre clientes por meio da escrita e leitura com pipes.
- Os arquivos e diretórios gerados pelo programa são criados no diretório /tmp e possuem uma hierarquia específica.

Hierarquia dos Arquivos

1. É criado o diretório /tmp/mac0352-11795888
    a. Chamarei de diretório raiz do projeto.
    b. Nesse diretório estarão contidos todos os outros arquivos e diretórios do EP.
2. Dentro do diretório raiz do projeto, cada tópico possuirá um diretório com seu nome.
3. Dentro do diretório de cada tópico, estarão todos os pipes que realizam comunicações para aquele tópico.

Nomes aleatórios para pipes

Os nomes dos pipes são gerados aleatoriamente, consistindo em uma sequência de 10 caracteres alfanuméricos.

Ou seja,

26 letras minúsculas + 26 letras maiúsculas + 10 números = 62 caracteres distintos

Logo, podem formar $62^{10} \sim 8.4 \times 10^{17}$ combinações únicas.

A probabilidade de repetição muito pequena.

Portanto, imaginando que há 3 clientes assinando o tópico **home**, seus pipes estarão localizados em:

/tmp/mac0352-11795888/home/S7pjN2wvP5
/tmp/mac0352-11795888/home/xsJREPA7Fr
/tmp/mac0352-11795888/home/PYY5WW7ZTH

E se tivermos mais 2 clientes assinando o tópico **work**, seus pipes estariam localizados em:

/tmp/mac0352-11795888/work/Yrm8jKG4om
/tmp/mac0352-11795888/work/gfeFW20WVj

Alterações na main

No arquivo oferecido pelo professor, realizei algumas mudanças significativas:

- Removi tudo aquilo que não me seria útil, como:
  - Criação dos pipes do exemplo;
  - Todo o corpo de código no espaço destinado ao EP1;
  - Variáveis do exemplo inicial (recvline, meu_pipe, meu_pipe_fd e outros)
- Incluí a minha biblioteca de funções auxiliares para o EP;
- Antes do loop infinito, a função `createRootDirectory` cria o diretório raiz do projeto em /tmp.
- O loop lê o primeiro byte do pacote recebido e joga para o parser da biblioteca fazer o procedimento adequado.

Interpretando comandos MQTT

O primeiro passo do broker para identificar um comando é ler o primeiro byte do pacote recebido pelo processo, e em seguida ele é passado para um parser.

Nos 4 primeiros bits do primeiro byte, há um identificador que diz qual é o tipo de pacote.

A partir do primeiro byte, o parser determinará qual será o caminho tomado pelo código, e o que será feito por aquele determinado processo naquele momento.

10: CONNECT

Caso o primeiro byte do pacote seja 10, o broker se encarrega de enviar de volta para o socket do cliente um pacote CONNACK, isto é, um pacote de comprimento fixo com os seguintes bytes:

`20 02 00 00`

Isto porque o CONNACK é o pacote enviado pelo broker ao cliente que **confirma a conexão bem sucedida.**

Ou seja, o cliente enviou um CONNECT para solicitar uma conexão com o broker, e somente isso.

30: PUBLISH

O PUBLISH é o pacote de envio de mensagem por parte dos publishers para algum tópico específico.

Caso o broker tenha recebido um pacote cujo primeiro byte é 30, trata-se de um pacote PUBLISH, e o trabalho do broker é **encaminhá-lo para os clientes** inscritos no tópico mencionado no pacote.

O processo que o código faz é descobrir o comprimento total do pacote, para escrever a quantidade correta de bytes nos pipes, e descobrir o nome do tópico, para escrever somente nos pipes dos subscribers que assinam aquele tópico.

A estrutura de um pacote PUBLISH (cujo QoS é zero) é:

O primeiro byte é sempre 30 (identificador do pacote PUBLISH), em seguida, temos de 1 a 4 bytes indicando o comprimento restante do pacote.

Depois 2 bytes com o comprimento do tópico, e, por fim, as strings do tópico e da mensagem enviada.

O broker é capaz de ler essas informações para escrever corretamente e encontrar os pipes corretos.

82: SUBSCRIBE

Caso o primeiro byte do pacote seja 82, trata-se de um pacote SUBSCRIBE, que solicita ao broker que faça o cliente assinar um determinado tópico.

O papel do broker neste caso é criar o pipe correspondente para aquele subscriber, enviar a confirmação de inscrição no tópico e iniciar o loop de leitura do pipe.

O broker lê a string do tópico incluída no pacote SUBSCRIBE, e cria um novo pipe para o processo do subscriber na pasta do tópico no diretório raiz do projeto.

Porém, a implementação deste comando possui sutilezas.

A principal é a de que, ao entrar no loop infinito, o processo ainda precisa ser capaz de receber pacotes do cliente. **Por isso, o loop de leitura do pipe roda em um fork.**

Ou seja, para cada cliente subscriber, há 2 processos no broker.

Por fim, é preciso enviar um pacote SUBACK para o cliente, cujo primeiro byte é 90, **como resposta à sua inscrição.** Da seguinte forma:

```
90 00 XX XX YY
```

Em que XX XX corresponde aos bytes do Packet Identifier recebidos no SUBSCRIBE.

E YY ao código de retorno do SUBSCRIBE. Será 00 se o cliente assinou o tópico com sucesso, e 80 se falhou, como no caso de não ser possível criar o pipe por algum motivo.

0C: PINGREQ

É um pacote simples enviado pelo cliente pedindo ping.

O broker sempre envia de volta ao cliente um pacote PINGRESP fixado como:

0d 00

Foi preciso ler os pipes do subscriber em um fork para que o processo pudesse responder aos pings.

0E: DISCONNECT

Por fim, o DISCONNECT é um pedido de desconexão do cliente ao broker.

O que o broker faz é:

1. Encerrar o processo de fork que lê o pipe em loop infinito, se houver algum.
2. Apagar o pipe atrelado ao processo, se houver algum.
3. Fechar a conexão com o socket.

Assim todos os processos são encerrados e o cliente é desconectado com sucesso.

Experimentos

Os experimentos a seguir foram realizados na minha rede doméstica em **duas máquinas diferentes.**

Meu laptop com ArchLinux rodava o broker, enquanto isso meu desktop, com Ubuntu WSL 2, executava todos os clientes mosquitto que se comunicavam com o laptop pelo IP e porta correspondente.

Nenhum cliente foi executado no localhost.

Todos os softwares que acessam a internet foram mantidos fechados durante o experimento, em ambas as máquinas.

Como foi realizada a conexão?

A conexão entre as máquinas foi realizada de modo que mantive a porta 1883 aberta em meu laptop.

A partir do desktop, conectei-me ao laptop por meio de seu IP privado, passado como parâmetro nos comandos `mosquitto_sub` e `mosquitto_pub`.

Medindo o uso de CPU

A medida do desempenho em termos da CPU foi dada de forma que a cada 0.5s retirava-se uma amostra do ps.

De cada amostra, somava-se o uso da CPU de todos os processos (forks) atribuídos ao broker.

E, a partir desses valores, pode-se construir uma análise mais robusta, avaliando uso médio de CPU, desvio padrão e seu comportamento no decorrer do tempo.

Scripts

Rodo o script a seguir para capturar saídas do comando ps a cada 0.5s de execução do broker.

```
while [ 1 ];
do
    echo "--------";
    ps ux | grep ep1 | grep -v grep;
    sleep 0.5;
done > /tmp/saida-cpu
```

Inicio no PUBLISH do primeiro cliente e interrompo no PUBLISH do último, ou seja, durante toda a captura houve o envio de pacotes por mosquitto_pub e recebimento por mosquitto_sub.

O comando anterior me entregará todas as saídas do programa ps divididos pela linha “--------”.

A terceira coluna dessas tabelas é o que me interessa, pois é onde estão contidas as porcentagens do uso de CPU por processo.

Para **eliminar as colunas adicionais**, rodei o comando:

```
awk ‘{print $3}’ /tmp/saida-cpu > /tmp/saida-cpu-porcentagens
```

Assim, ficamos com um arquivo somente com os valores das porcentagens de cada captura, sendo cada captura separada por uma linha vazia.

Antes de somar todos esses valores, desejo separá-los para encontrar o uso de cpu total do broker a cada momento do tempo. Posso separar cada sequência de valores pelo comando:

```
awk -v RS= '{print > ("saida-cpu-porcentagens-" NR)}'
/tmp/saida-cpu-porcentagens
```

A linha acima criará diversos arquivos, um para cada momento no tempo, com os valores das porcentagens de cada processo em cada linha.

Por fim, basta agora somarmos os números de cada arquivo, e teremos em cada um deles o uso total de CPU pelo processo do broker naquele instante no tempo.

Basta rodar para cada arquivo o comando:

```
paste -s -d '+' saida-cpu-porcentagens-X | bc -l
```

Em que X é o instante do tempo desejado.

Medindo a carga na rede

A carga na rede foi medida através do wireshark.

Foi aplicado o filtro de captura `tcp port` 1883, portanto, a captura foi restringida para a porta padrão do broker MQTT.

A análise da carga na rede é dada de forma que observamos a quantidade de bytes transferidos nos dois sentidos, e o tempo necessário para que essa transmissão e recepção acontecesse.

Observação importante!

O broker, ao lidar com forks e pipes, não é tão escalável quanto eu gostaria para esses experimentos.

Após o SUBSCRIBE dos subscribers, é preciso aguardar um **cooldown** de alguns segundos antes de enviar PUBLISH por meio dos publishers. Caso contrário, alguns processos de subscribers não terão tido tempo suficiente para criar seus pipes, e não receberão os pacotes desejados.

A criação dos pipes e forks fica muito lenta conforme mais clientes são conectados.

As medições dos experimentos foram realizadas após o SUBSCRIBE de todos os mosquitto_sub, para que o recebimento da mensagem pudesse ser garantido.

Nenhum cliente conectado

Este é o resultado trivial.

- Não há muito o que dizer.
- Uso da CPU no htop e no ps é tão baixo que é impresso como 0.0
- Não há nada na rede. Nenhum pacote é enviado nem recebido, como esperado.
- Há somente um processo sendo executado do início ao fim.

100 subscribers e 100 publishers

Dessa vez, foram conectados 100 clientes mosquitto_sub e depois 100 clientes mosquitto_pub que enviaram 100 pacotes PUBLISH.

Foram criados 5 tópicos: home, work, farm, jungle, church.

Cada tópico possui 20 subscribers inscritos, totalizando 100.

E foram realizadas 20 iterações de publish, em cada uma delas eram enviadas 5 PUBLISH, um para cada um dos 5 tópicos.

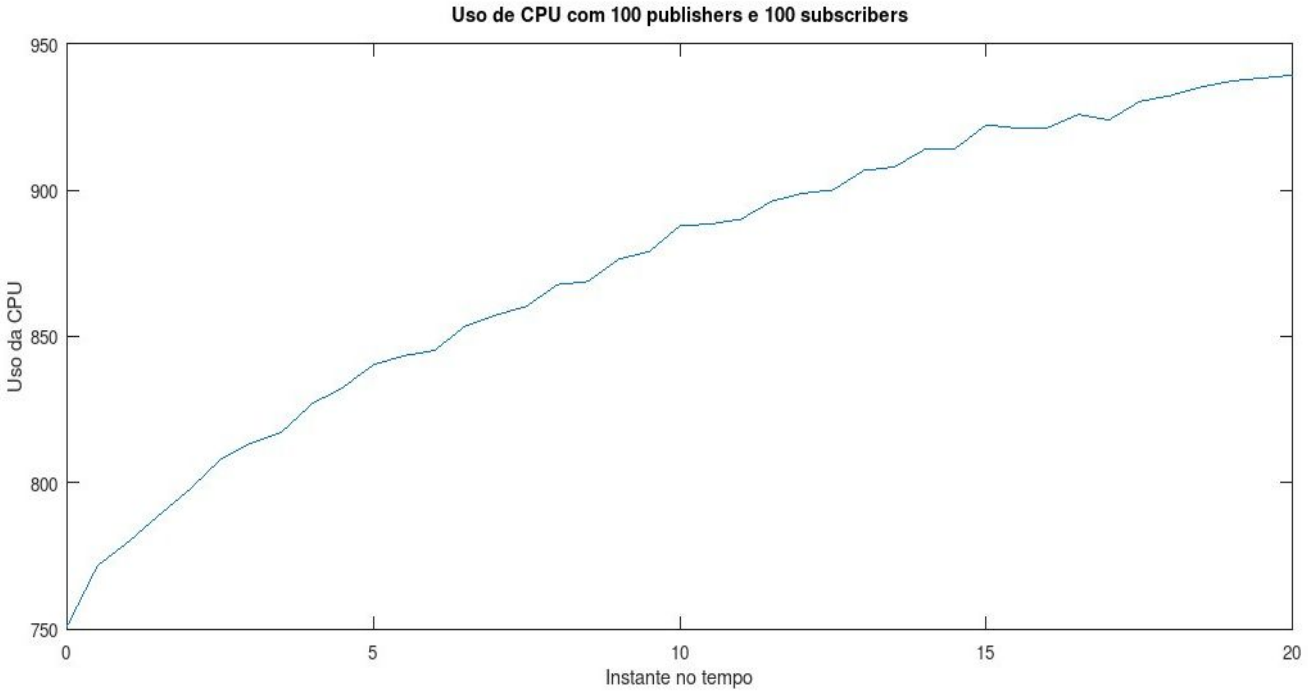
O experimento foi bem sucedido, e foi realizado dezenas de vezes para a interpretação dos dados a seguir.

Porém, desde o princípio, independentemente da quantidade de clientes conectados, vale notar que os processos consomem 90%+ de CPU, ao menos o que consta em programas como htop.

Apesar de procurar, não compreendi muito bem essa situação, e, como era o que tinha documentado, foi o que utilizei para geração dos dados.

Média: 873.419512

Desvio padrão (não diz muito) : 52.285276



1000 subscribers conectados e 1000 publishers

Infelizmente não foi possível executar o broker com essa escala na minha máquina.

O programa foi capaz de conectar os subscribers. Porém, usava a CPU em nível crítico 99%+ desde o início, e, conforme os clientes se conectavam, ficava cada vez mais lento para enviar um PUBLISH, igualmente ao enviar mais mensagens.

Encerrei no ponto de 1000 subscribers e 80 publishers, em que o PUBLISH já estava demorando tanto tempo que não conseguiria terminar o experimento no prazo.

Certamente demoraria alguns dias, se o computador não desligasse sozinho.