

MOTT

EP1 - MAC0352 - Redes de Computadores e Sistemas Distribuídos
Daniel Silva Lopes da Costa - 11302720

Message Queuing Telemetry Transport

- É um protocolo de mensagens leve para sensores e pequenos dispositivos móveis otimizado para redes TCP/IP.
- É um protocolo da camada de aplicação projetado para um baixo consumo de banda de rede e recursos de hardware, desenvolvido pela IBM e Eurotech na década de 90.
- Os princípios arquitetônicos são minimizar o uso de banda de rede e uso de recursos dos equipamentos enquanto garantindo confiabilidade e algum nível de garantia de entrega.
- MQTT hoje é usado em uma ampla variedade de indústrias, como automotiva, manufatura, telecomunicações, petróleo e gás, etc.
- Nesse exercício programa foi implementada a versão 3.1.1 do broker MQTT na linguagem C.
- É um protocolo que usa a estrutura padrão Publisher/Subscriber.

Pacotes MQTT

MQTT define pacotes de controle (Control Packets) para indicar a ação desejada a ser executada pelo recurso desejado. O MQTT possui uma estrutura de pacotes de controle própria, que basicamente é constituída de três partes principais: Cabeçalho fixo, cabeçalho variável e o payload.



Pacotes MQTT



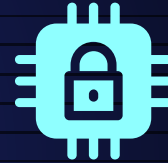
Cabeçalho fixo

O primeiro Byte contém o tipo do pacote e algumas flags de controle. O segundo Byte expressa o número de bytes restante no pacote.



Cabeçalho variável

Tamanho varia de acordo com o tipo do pacote, contém identificadores do pacote.



Payload

Onde temos a mensagem em si, geralmente o tópico ou mensagens para a troca de informações.



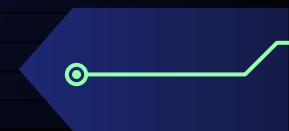
Detalhes de IMPLEMENTAÇÃO

Detalhes de implementação.

- Como já citado nesse exercício programa foi implementada a versão 3.1.1 do broker MQTT na linguagem C.
- Foi utilizada uma estrutura de divisão de processos por meio da função **fork()**, cada cliente ficava em um processo diferente de forma paralela.
- Também foi utilizado pipes nomeados, para fazer a troca de mensagens, cada subscriber possui um pipe que fazia referência a uma tópico. Quando o publisher envia uma mensagem sobre um tópico, o broker passa por todos os pipes nomeados relacionados a esse tópico.
- Para a criação dos pipes foi utilizada a função **mkfifo()**, que cria uma arquivo temporário. O nome do arquivo foi determinado pelo número de processo do subscriber, identificado pelo **getpid()**.



Detalhes de implementação.

- Além disso, foi usada a função **mmap()**, para alocar memória necessária guardar algumas informações que precisavam ser comuns a todos os processos como o identificador de cada tópico e quais pipes(processos) estavam inscritos em um tópico;
 - A principal estrutura de dados utilizada para salvar as informações foram listas com tamanho pré-definido;
 - Foi utilizado como base o arquivo echo disponibilizado pela disciplina, adicionando as estruturas necessárias para o funcionamento do broker, mas a estrutura dos soquetes do servidor, permaneceu inalterada.
- 

PACOTES TRABALHADOS



CONNECT

Cliente solicita uma
ligação com um servidor



CONNACK

Reconhece solicitação de
conexão



PUBLISH

Publicar mensagem sobre
um determinado tópico



PINGREQ

PING request



SUBSCRIBE

Inscrever-se em um
tópico



SUBACK

Reconhecimento de
inscrição

CONNECT / CONNACK

Ao analisar o pacote enviado pelo cliente, no primeiro byte é feito um shift de quatro bits para a esquerda, isso resulta no tipo do pacote. No CONNECT temos o valor 1 como identificador do tipo de pacote. Ao receber o CONNECT, o broker cria um pacote CONNACK, para reconhecer a conexão. É um pacote de 4 bytes, onde o payload é nulo.



PUBLISHER

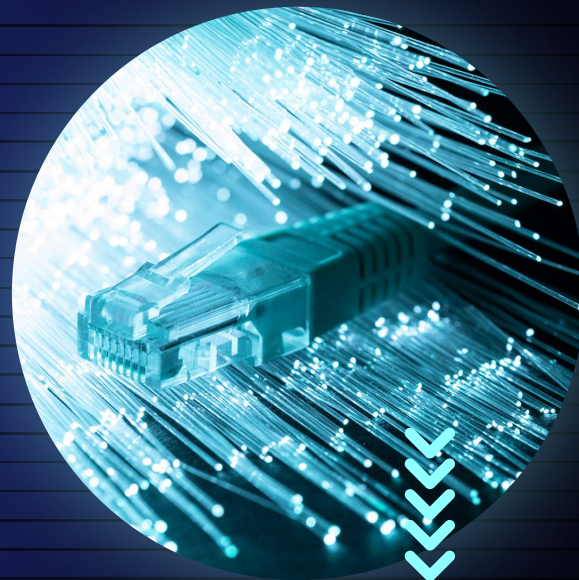
É realizado o mesmo processo para identificar o tipo de pacote. Em seguida, é feita a leitura do terceiro e quarto bits que são identificadores da mensagem. No quinto e sexto tem-se o tamanho do tópico. Na sequência lemos o tópico com base no tamanho encontrado. Por fim, usa-se o tamanho no remaining length para encontrar o tamanho da mensagem.

Com isso, identificamos se aquele tópico possui algum subscriber e iteramos por eles, escrevendo a mensagem no Pipe de cada cliente.



SUBSCRIBER / SUBACK

No Subscriber, inicialmente identifica-se o tópico para o qual ele que se inscrever, se o tópico já existe é passado o identificador, do contrário é criado um estrutura para armazenar o novo tópico. Em seguida é criado o SUBACK, se não houver espaço para um novo tópico é enviado um suback informando erro, do contrário temos uma mensagem de sucesso. Com isso é criado um novo arquivo, que terá o nome formado pela string “/tmp/temp.mac0352.xxxxxx” onde xxxxxx = PID do processo. Em uma lista, no índice correspondente ao PID adicionamos o identificador do tópico. Por fim, abrimos um loop infinito, onde o processo fica esperando por atualizações no arquivo.



PINGREQ / PINGREQ

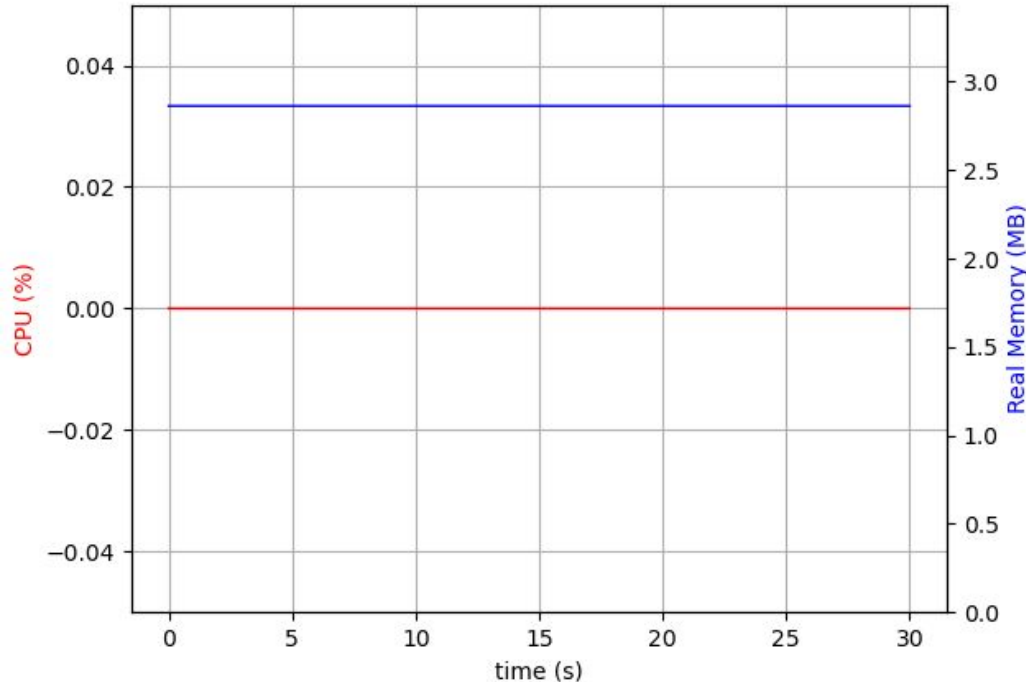
Recebe pacote, com identificador 12, faz o encode do PINGREQ que no caso são apenas dois bytes, não possui payload:

PINGREQ = [0xd0, 0x00]



Análise dos Testes

BROKER SOZINHO



0%

CPU

Provavelmente o uso de CPU foi tão baixo que ficou como zero.

2.8MB

Memória

Essa memória acredito que seja a do malloc inicial que reserva memória para armazenar os tópicos e clientes.

1 SUBSCRIBER x 1 PUBLISHER

~120%

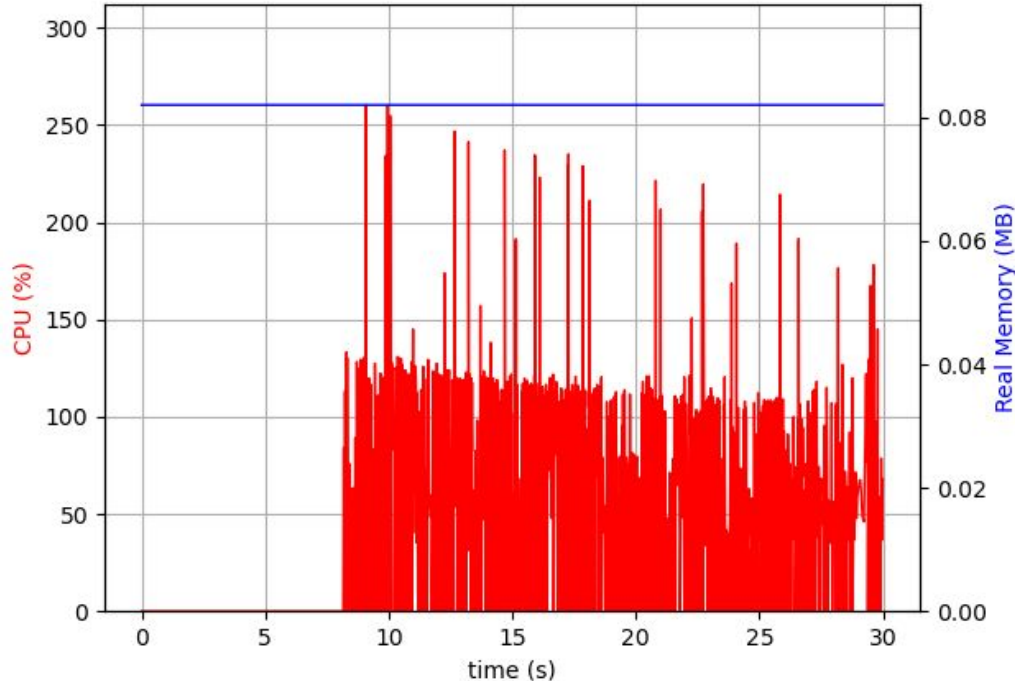
CPU

Uso da CPU dá um salto quando a mensagem do pub é enviada um pouco antes dos dez segundos.

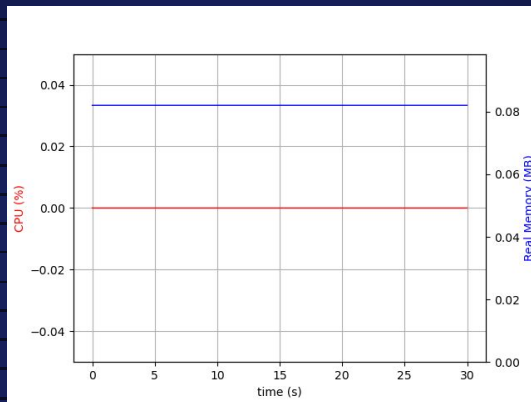
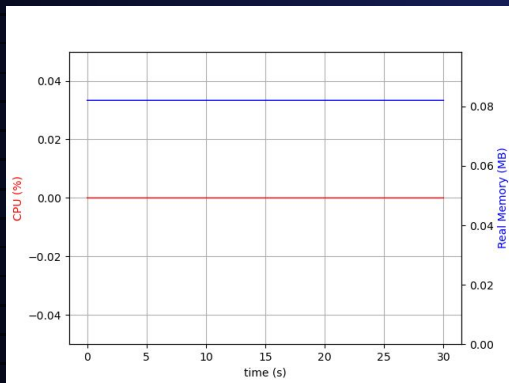
3 MB

Memória

Memória aumentou pouco, provavelmente por conta de alguns mallocs locais

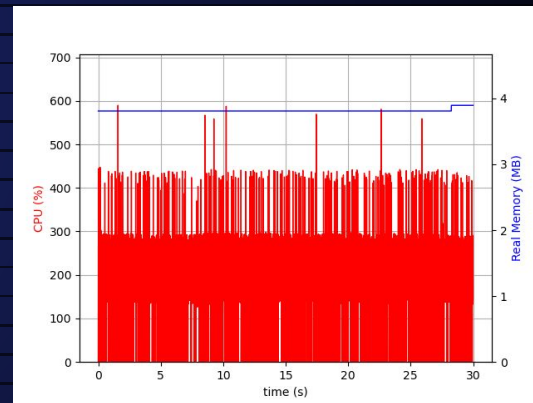


1 SUBSCRIBER
X
0 PUBLISHER

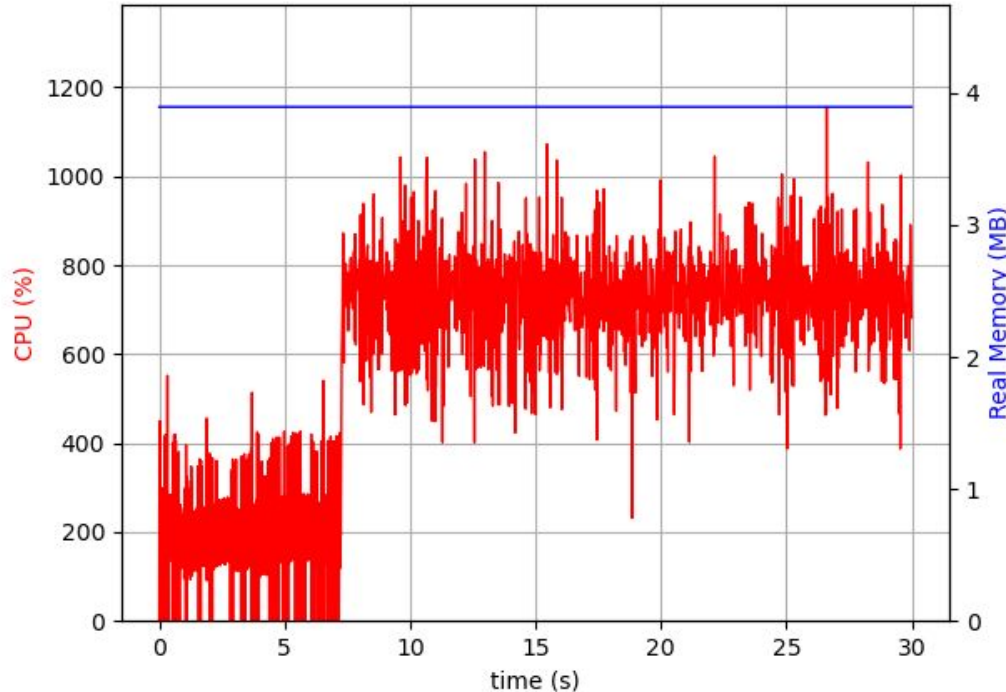


2 SUBSCRIBER
X
0 PUBLISHER

10 SUBSCRIBER
X
0 PUBLISHER



10 SUBSCRIBER x 1 PUBLISHER



800%

CPU

Novamente o uso de CPU aumentou consideravelmente com o envio da mensagem.

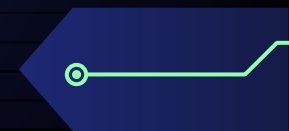
3.9MB

Memória

Novamente acredito que o acréscimo de memória seja de mallocs locais em cada um dos subscribers.



Testes - Análise de resultados

- Sobre os gráficos vale destacar os saltos no uso de CPU quando era publicada uma mensagem, isso provavelmente ocorre pois a mensagem passa por um loop de 500000 para ver se tem algum pid escrito no tópico, isso não é muito eficiente e está provocando esse grande aumento no processamento.
 - A memória também deve ser proveniente dessa estrutura, pois no início do programa é feito um malloc global de tamanho meio milhão.
 - Os testes foram realizados com o uso de uma outra máquina com oito núcleos de CPU dedicados e 16G de memória RAM, e uma máquina host com as mesmas características.
 - A comunicação foi feita utilizando uma rede local de 150 Mbps;
- 

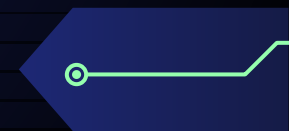
Testes - Análise de resultados

- Não consegui implementar nenhuma estrutura automatizada para fazer os testes com 100 e 1000 clientes, portanto reduzi minha análise a casos implementados manualmente;
- Aconteceram algumas inconsistências, muito provavelmente porque os teste foram executados poucas vez;
- Apesar disso, acredito que experimentos foram bem sucedidos e servem de referência sobre o uso de CPU e memória para o broker MQTT desenvolvido.
- Para geração dos gráficos foi utilizada a biblioteca psrecord, onde é passado um processo e a biblioteca gera os gráficos do processo e também dos subprocessos filhos, analisando consumo de memória de CPU.



PROBLEMAS



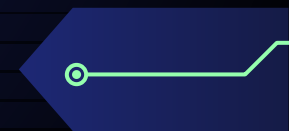


- Perdi muito tempo por não saber o formato da mensagem que seria enviada para o publisher, após o envio pelo subscriber, por algum motivo assumi que seria apenas a mensagem em ASCII, por isso na decodificação do pacote do publisher separei a mensagem. No final consegui descobrir que mensagem deveria ser enviada em completo para o subscriber.
 - Outro ponto negativo foi que apenas usei lista para armazenar os tópicos e os subscribers o que deixou a solução não escalável e ineficaz - tentei implementar as estruturas com lista ligadas, mas por algum motivo, mesmo fazendo o malloc global, quando mudava de processo, o arquivo salvo no nó apresentava inconsistências - esse foi outro motivo que me levou a uma grande perda de tempo.
 - Outro problema é que não estou conseguindo enviar mensagens para um mesmo tópico de maneira instantânea. Se uma mensagem for enviada, tem que esperar aproximadamente 60s para enviar outra mensagem, do contrário a mensagem não vai chegar nos subscribers.
- 



CONCLUSÕES


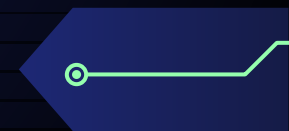


- Temos como resultado o broker mqtt solicitado, com algumas inconsistências, mas funcionando corretamente e realizando o processo esperado de envio de mensagens;
 - Muitos aprendizados sobre a leitura de RFCs e especificações sobre protocolos de rede;
 - Com isso podemos concluir que tivemos uma implementação bem sucedida do Broker MQTT, realizando as trocas de mensagens corretamente e garantindo a experiência necessária para o desenvolvimento de um servidor de rede.
- 
- 
- 



REFERÊNCIAS



- <https://pt.wikipedia.org/wiki/MOTT>
 - http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html#_Toc442180876
 - <https://docs.pipz.com/central-de-ajuda/learning-center/guia-basico-de-markdown#open>
 - <https://www.delftstack.com/pt/howto/c/how-to-convert-an-integer-to-a-string-in-c/>
 - <https://mqtt.org/>
 - <https://github.com/astrofrog/psrecord>
- 
- 

The background is a dark blue gradient with various light blue and white decorative elements. These include circuit-like lines, arrows pointing in different directions, and clusters of small dots. The word 'OBRIGADO!' is centered in a large, bold, white font.

OBRIGADO!

EP1 - MAC0352 - Redes de Computadores e Sistemas Distribuídos
Daniel Silva Lopes da Costa - 11302720