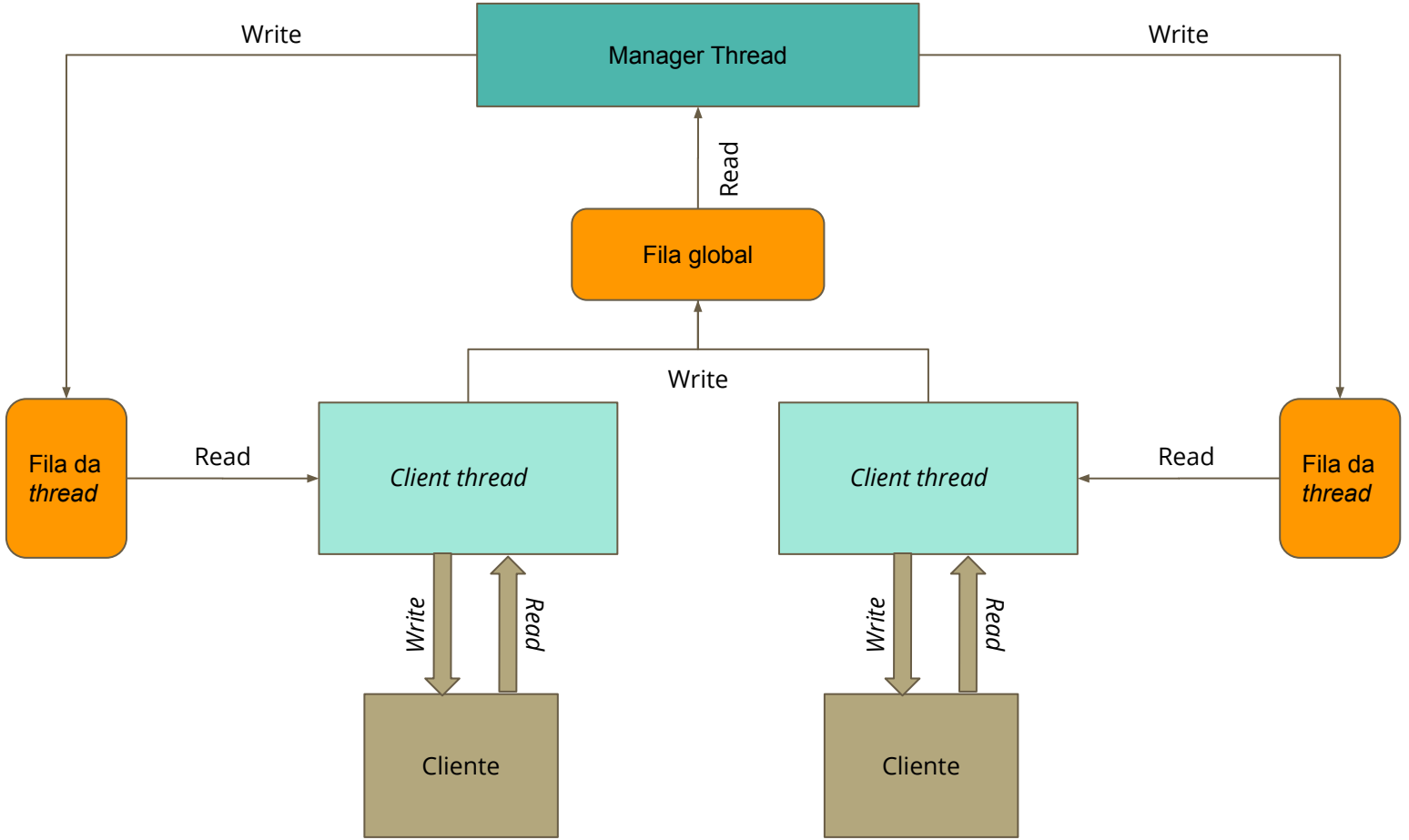
EP1 - Redes de Computadores e Sistemas Distribuídos (2021)

Thiago Cunha Ferreira

Esquema geral do projeto

- *Broker* consiste de duas seções diferentes:
  - Um “processo” *manager* que realiza a lógica dos comandos vindos dos clientes (com o propósito final de enviar mensagens para os clientes corretos)
  - Diversos “processos” *clients* responsáveis por fazer a comunicação direta, tanto na leitura quanto escrita, dos clientes com o *broker*. Um por cliente.
- O servidor foi implementado utilizando *threads*, com *manager* e *clients* sendo *threads* distintas.
- A comunicação se dá exclusivamente entre *manager e client* e é feita através de filas de mensagens (cada *thread possui uma*).

Esquema geral



Sobre a *client thread*

A *client thread* trabalha da seguinte forma:

1. Espera por pacote vindo da conexão com o cliente ou mensagem vindo da *manager thread*.
2. **No primeiro caso**, geralmente monta-se uma estrutura contendo a informação do pacote a partir da sequência de bytes recebidos e manda, com outras informações, referências para a *manager thread*.
3. **No segundo caso**, obtém-se a estrutura enviada no passo 2 (com possíveis alterações) e informações adicionais para poder então montar uma mensagem para ser enviada de volta a um cliente.
4. Caso uma mensagem do tipo **DISCONNECT** vinda da *manager thread* apareça (ou a conexão fecheabruptamente), destrói-se a *thread* (não antes de ordenar as estruturas na *manager thread* relacionado à *client thread* a ser extinta). Caso contrário, volta ao passo 1

Um detalhe de implementação notório é que, para podermos esperar por dados vindos tanto do *socket* de conexão quanto da fila de mensagens (estrutura de dados interna ao programa), utiliza-se um *pipe* onde se é escrito 1 *byte* sempre que uma mensagem for disponibilizada.

Sobre a *manager thread*

A *manager thread* trabalha da seguinte forma:

1. Fica inativa até haver uma mensagem vinda da fila de mensagens global, onde todas as *client threads* publicam mensagens. Quando há alguma mensagem, pega a primeira da fila.
2. Processa-se a mensagem vinda de uma *client thread*, geralmente constituídas de uma série de ponteiros para informações relevantes à requisição. Com isso, uma série de operações sobre estruturas locais são feitas.
3. Publica-se uma mensagem na fila de mensagens da *client thread* associada à mensagem processada.
4. Volta ao passo 1

A *manager thread* mantém algumas estruturas internas utilizadas para a lógica associada ao funcionamento do broker MQTT. Elas são:

- Uma lista (finita) de conexões, contendo referência a informações sobre um cliente/*client thread* e uma lista (finita) de ponteiros para inscrições
- Uma lista (finita) de tópicos, composta de um nome e uma lista (finita) de espaços disponíveis (*subscriptions*).
- *Subscriptions*, que nada mais são que referências ao nome do tópico e a uma conexão.

Sobre comandos - CONNECT / CONNACK

Apesar de haver muitas configurações embutidas num pacote CONNECT que apontam o funcionamento esperado do servidor pelo cliente (sendo que a maior parte não foi implementada), o principal elemente associado ao pacote é o "*client ID*". Ele é quem irá identificar, no lado da *manager thread*, a qual *client thread* / conexão a mensagem está se referindo.

No lado da *client thread*, assim como a maioria dos comandos implementados, pacotes CONNECT passam por um processamento, são transformados em uma struct e passados para a *client thread*. Quando chega uma mensagem CONNACK vinda da *manager thread*, utiliza-se essa struct para formar uma mensagem do tipo CONNACK para o cliente.

No lado da *manager thread*, procura-se um *slot* onde podemos depositar um ponteiro para informações como:

- Endereço da fila de mensagens da *client thread*
- *Pipe* usado para alertar a *client thread* que uma mensagem da *manager thread* está disponível
- *Client ID*

Sobre comandos - SUBSCRIBE / SUBACK

- O comando de subscribe é um dos mais complexos de se processar no lado da *manager thread*, pois envolve alterar muitos ponteiros em relações circulares pouco convencionais e múltiplas checagens por posições livres em arrays.
- Assim como no caso do CONNECT / CONNACK, há muitas opções de interações especificadas no protocolo, porém a maior parte deles não está implementada. Entretanto, uma coisa que se permite fazer são múltiplas *subscriptions* pelo mesmo pacote.

Sobre comandos - PUBLISH

- Como pacotes do tipo PUBLISH (aqui implementando) só envolvem, basicamente, reenviar o mesmo pacote para outros clientes (possivelmente com pequenas alterações), considerando que esses pacotes podem chegar a tamanhos bem grandes (limites máximos do MQTT) e já que podemos compartilhar memória alocada no *heap* entre as *threads*, o que se faz é passar para a *manager thread* uma mensagem com um ponteiro para o pacote inteiro (além de outras coisas).
- Ao invés de ficar acompanhando quem está escrevendo o pacote PUBLISH para qual cliente, deixamos que a desalocação da mensagem fique por conta do conjunto de *client threads*, de forma distribuída. Isso é feito com um contador e um mutex.

Sobre comandos - PINGREQ / PINGRESP e DISCONNECT

- Devido à simplicidade dos pacotes PING, eles são tratados e respondidos sem passar pela *manager thread*
- Pacotes DISCONNECT, no lado da *client thread*, são bem simples também. Porém, quando um cliente irá se desconectar, é necessário remover um conjunto de referências associadas ao cliente que ficam na *manager thread*. Isso geralmente envolve a anulação de diversos ponteiros e liberação da estrutura primária de identificação de pacotes.