MC714 - 1º Sem. 2024

Trabalho 2

Exemplo de sistema distribuído
utilizando conceitos de

- Relógio de Lamport
- Exclusão Mútua
- Eleição de Líder

Tiago Godoi Bannwart
215386

1. Visão Geral

O sistema consiste de 4 componentes que rodam simultaneamente. Cada um está num container Docker isolado, e trocam mensagens através de um broker MQTT. A linguagem escolhida foi TypeScript.

O objetivo do sistema se resume a decidir quem é o líder, checar quem está online e acessar um recurso hipotético, um por vez. Tudo isso podendo saber quais mensagens vieram antes nos casos de mensagens simultâneas.

2. Descrição dos arquivos

```
.
├── README.md
├── component
│   ├── Dockerfile
│   ├── bun.lockb
│   ├── package.json
│   ├── src
│   │   ├── device.ts
│   │   ├── index.ts
│   │   └── util.ts
│   └── tsconfig.json
└── docker-compose.yml
```

index.ts:	Arquivo principal, instancia a classe Device e inicia a eleição e o token-ring para exclusão mútua.
device.ts:	Contém a declaração da classe Device e toda a lógica do relógio de Lamport, exclusão mútua e eleição.
util.ts:	Contém funções auxiliares.
package.json:	Registra as dependências.
Dockerfile:	Descreve o container de cada dispositivo.
docker-compose.yml:	Define quantas instâncias do container devem ser executadas.

3. Implementação

3.1 Bibliotecas

A única biblioteca utilizada foi ‘mqtt’, que auxilia no processo de conectar no broker.

3.2 Sistema de Comunicação

O método de troca de mensagens escolhido foi o MQTT por sua simplicidade. Ao iniciar, cada componente se conecta no broker *“test.mosquitto.org”* e se inscreve no tópico ‘mc714-tgb', onde todas as mensagens são publicadas. Todas as mensagens são formatadas em JSON e seguem o seguinte formato:

```
{
  sender: number; // ID do rementente
  to: number;     // ID do destinatário (-1 para todos os dispositivos)
  data: string;   // Mensagem em si
  clock: number;  // Relogio local do rementente
}
```

A classe Device tem duas funções para troca de mensagens: send() e receive().

3.3 Relógio de Lamport

A implementação do relógio de Lamport foi feita da seguinte forma:

- A função send() incrementa o relógio local em +1, anexa ele na mensagem e a envia.
- a função receive() lê o relógio que está contida na mensagem e atualiza o relógio local buscando o maior entre os dois, e depois somando +1.
- a função print(), implementada em util.ts, sempre coloca na frente de cada texto impresso no terminal o tempo em que cada evento aconteceu em cada componente.

Os exemplos apresentados nas próximas partes ilustram isso melhor.

3.4 Eleição

O método escolhido para a eleição foi o ‘Bully’ por conta de sua versatilidade, e porque a rede não precisaria seguir uma topologia específica. O cenário simulado pelo programa é o seguinte:

- Um dos componentes inicia uma eleição, enviando ‘Eleicao’ para todos, a fim de descobrir quem é o líder.
- Caso não receba resposta de ninguém após 3 segundos, se auto declara líder e avisa todos enviando ‘ImLeader’.
- Quando outro componente recebe a mensagem de eleição, respondem ‘ImHigher’ apenas se tiverem um ID maior do que o remetente. Nesse caso, eles aguardam alguns segundos e iniciam a própria eleição.
- Se o componente que iniciou a eleição receber a mensagem ‘ImHigher’ ele desiste de ser líder.
- Todos que receberem ‘ImLeader’ registram aquele como o novo líder.

Componente 0	Componente 1	Componente 2	Componente 3
1. Sent: Election	1.	1.	1.
2.	2. from 0: Election	2. from 0: Election	2. from 0: Election
3.	3. Sent: ImHigher	3. Sent: ImHigher	3. Sent: ImHigher
4. from 1: ImHigher	4. Sent: Election	4. Sent: Election	4. Sent: Election
5. from 1: Election	5. from 2: Election	5. from 1: Election	5. from 1: Election
6. from 2: ImHigher	6. from 3: Election	6. Sent: ImHigher	6. Sent: ImHigher
7. from 2: Election	7. from 2: ImHigher	7. from 3: Election	7. from 2: Election
8. from 3: ImHigher	8. from 3: ImHigher	8.	8. Sent: ImHigher
9. from 3: Election	9.	9. from 3: ImHigher	9. Sent: ImLeader
10. from 3: ImLeader	10. from 3: ImLeader	10. from 3: ImLeader	

1. (Verde) Componente 0 inicia a eleição e perde
2. (Azul) Componente 1 inicia a eleição e perde
3. (Vermelho) Componente 2 inicia a eleição e perde
4. (Amarelo) Componente 3 inicia a eleição, ganha e avisa os outros

Os números na esquerda da simulação representa o relógio de Lamport de cada componente.

3.5 Exclusão Mútua

Para implementar uma exclusão mútua foi escolhido o método Token-Ring. Toda vez que um componente recebe o token, que representa a permissão para acessar um recurso, ele faz o acesso (na prática, ele só espera alguns segundos) e repassa para o próximo componente.

Para que esse algoritmo funcione, é preciso que todos saibam quais componentes estão online. Para isso, o líder manda uma mensagem ‘WhoIsOnline’, e todos que a receberem respondem ‘ImOnline’. Dessa forma, é possível criar uma lista de quem está online.

O anel então segue a ordem crescente de dispositivos, começando no componente 1 (escolhido arbitrariamente), depois 2 -> 3 -> 0 -> 1, iniciando de novo o circuito.

O token é repassado apenas enviando a mensagem ‘Token’ para um destinatário específico.

4. Simulação

Por motivos de simplicidade, o programa não fica realmente checando se o líder está online e também não existe um recurso que é acessado durante a exclusão mútua. Dessa forma, o programa serve para simular um cenário específico e demonstrar o funcionamento dos algoritmos naquela situação. É possível escolher os seguintes parâmetros:

- Quantidade de componentes (modificando docker-compose.yml)
- Qual dispositivo inicia a eleição (em index.ts)
- Qual dispositivo inicia com o token (em index.ts)

Uma simulação com 4 componentes, em que o de ID 0 inicia a eleição, e ID 1 começa com o token se parece o com seguinte:

Componente 0	Componente 1	Componente 2	Componente 3
Eleição (ja explicado em 3.4)			
1. Sent: Election	1.	1.	1.
2.	2. from 0: Election	2. from 0: Election	2. from 0: Election
3.	3. Sent: ImHigher	3. Sent: ImHigher	3. Sent: ImHigher
4. from 1: ImHigher	4. Sent: Election	4. Sent: Election	4. Sent: Election
5. from 1: Election	5. from 2: Election	5. from 1: Election	5. from 1: Election
6. from 2: ImHigher	6. from 3: Election	6. Sent: ImHigher	6. Sent: ImHigher
7. from 2: Election	7. from 2: ImHigher	7. from 3: Election	7. from 2: Election
8. from 3: ImHigher	8. from 3: ImHigher	8.	8. Sent: ImHigher
9. from 3: Election	9.	9. from 3: ImHigher	9. Sent: ImLeader
10. from 3: ImLeader	10. from 3: ImLeader	10. from 3: ImLeader	
Busca de quem está online			
10.	10.	10.	10. Sent: WhoIsOnline
11. from 3: WhoIsOnline	11. from 3: WhoIsOnline	11. from 3: WhoIsOnline	11.
12. Sent: ImOnline	12. Sent: ImOnline	12. Sent: ImOnline	12.
13. from 2: ImOnline	13. from 2: ImOnline	13. from 1: ImOnline	13. from 2: ImOnline
14. from 1: ImOnline	14. from 0: ImOnline	14. from 0: ImOnline	14. from 1: ImOnline
			15. from 0: ImOnline
Token-Ring			
15.	15. Sent: Token →	15.	15.
16.	16.	16. from 1: Token	16.
17.	17.	17. Sent: Token →	17.
18.	18.	19.	18. from 2: Token
19.	19.	20.	19. Sent: Token
20. from 3: Token	20.	21.	
21. Sent: Token →	21.	22. from 0: Token	
		23. Sent: Token	

5. Considerações Finais

Apesar do programa não realizar nenhuma tarefa verdadeiramente útil, só a tarefa de alguns dispositivos diferentes estarem rodando simultaneamente e trocando mensagens já impõe alguns desafios encontrados em sistemas reais.

As soluções implantadas ilustram bem como que esses desafios são superados e permitem entender melhor como que acontece o processo, algo que geralmente não é tão simples.

Além disso, tendo essa base sólida implementada, designar uma tarefa de verdade para o sistema resolver não seria muito trabalhoso, ou seja, ele pode ser tornar útil para uma aplicação real.