

Restaurante Drive-Through

Peer-to-Peer

Trabalho de avaliação No. 1

Universidade de Aveiro
Licenciatura em Engenharia Informática
UC 40382 - Computação Distribuída
Abril de 2019 - 2º Semestre

Trabalho realizado por:

88808 - João Miguel Nunes de Medeiros e Vasconcelos
88886 - Tiago Carvalho Mendes

Código do projeto disponível em:



<https://github.com/detiuaveiro/drive-through-p2p-tiagocmendes>

1. Introdução

O presente documento tem como principal objetivo descrever os pontos principais da solução desenvolvida pelos alunos apresentados na capa do mesmo, tendo em conta o problema proposto como primeiro trabalho prático de avaliação da unidade curricular de Computação Distribuída da Universidade de Aveiro. Este problema consistia na implementação de um sistema distribuído, em uma estrutura de anel (*token-ring*), simulando as ações desempenhadas por diferentes entidades que constituem um restaurante. Para alcançar esse objetivo, o projeto foi desenvolvido na linguagem de programação *Python*.

Tendo em conta o problema proposto no enunciado disponibilizado, este dividia-se em duas grandes partes:

- A primeira fase, referente ao “Setup de simulação”, requeria que fossem inicializados diversos nós, cada um conhecendo apenas o seu sucessor, com o intuito de formar um anel consistente e pronto a comunicar. De seguida, deveria ser iniciado um processo de descoberta dos nós, de forma a garantir que todos os nós se conheciam uns aos outros.
- A segunda fase, referente à “Implementação da simulação”, requeria que fossem simuladas as ações desempenhadas pelas diversas entidades constituintes de um restaurante. Estas entidades seriam o próprio Restaurante, o Rececionista (Clerk), o Cozinheiro (Chef) e o Empregado de Balcão (Waiter).

2.1.1 Setup de Simulação - Criação do anel

Para a criação do anel, decidimos construir uma classe denominada **ringNode**, que implementa cada nó do mesmo. Cada um destes nós tem um nome, um identificador e um endereço próprios, caracterizando a respetiva entidade na simulação. A nossa solução pressupõe que, independentemente da ordem da inicialização destes nós, todos se conseguem juntar ao anel, e ordenar-se por ordem crescente de identificadores (ID). Ainda que o endereço de acesso ao anel (ou seja, o endereço para qual todas as entidades enviem um primeiro pedido de NODE_JOIN) possa ser o de qualquer um dos nós, uma vez escolhido tem de ser o mesmo para todos os nós que se queiram juntar ao anel. O nó cujo endereço seja o de acesso ao anel não tem necessariamente de ser o primeiro a ser inicializado embora, caso o seja, diminua o tempo de formação do anel. Uma das muitas possibilidades da criação do anel está exemplificada em baixo.

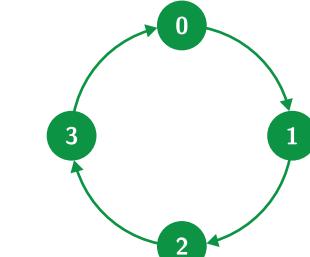
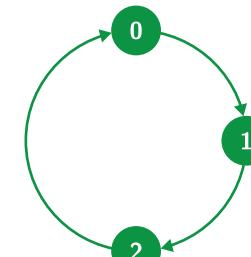
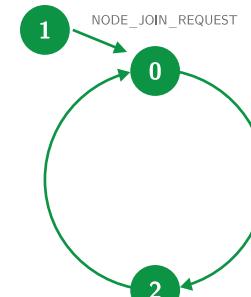
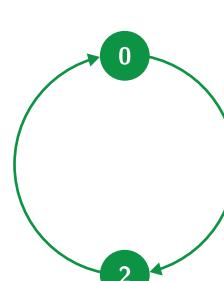


Figura 1
O sucessor da primeira entidade é ela própria

Figura 2
Primeiro pedido de junção ao anel

Figura 3
Anel com dois nós (entidades)

Figura 4
Anel com dois nós (entidades)

Figura 5
Anel com três nós (entidades)

Figura 6
Anel com quatro nós (entidades)

2.1.2 Setup de Simulação - Comunicação dentro do anel

Na sequência da página anterior, assim que o nó cujo endereço é o de acesso ao anel é inicializado, este envia uma mensagem para o seu sucessor com o método “RING_COUNT”, de modo a verificar se o anel já está completo. Sempre que uma entidade que não seja a do endereço de acesso recebe esta mensagem, incrementa o contador presente na mesma, reencaminhando-a de seguida para o seu sucessor. Quando a mensagem volta a chegar ao endereço inicial, este verifica se o contador é igual ao número de nós pretendido no anel. Em caso afirmativo, começa o processo de NODE_DISCOVERY. Caso contrário, reinicia o valor desse contador e continua o processo de verificação do tamanho do anel.

2.1.3 Setup de Simulação - Descoberta das entidades

Após o anel estar completo, é iniciado o processo de descoberta das entidades. Para tal, a entidade com o endereço de acesso ao anel envia um TOKEN para o seu sucessor com o método “NODE_DISCOVERY” e argumentos um dicionário cuja chave é o seu nome e o valor o seu ID ({'Nome': ID}). Cada uma das seguintes entidades recebe este dicionário, atualiza a sua tabela de entidades local e junta ao dicionário de entidades o seu nome e identificador, sucessivamente. No entanto, todas as entidades só vão conhecer, garantidamente, todas as outras quando o TOKEN passar pela entidade inicial 2 vezes. Quando isto acontecer, o processo de descoberta das entidades está completo e pode-se iniciar a simulação do restaurante.

2.1.4 Setup de Simulação - Implementação de cada entidade

Para implementarmos cada entidade, decidimos criar 4 classes diferentes chamadas de Restaurant, Clerk, Chef e Waiter, cada uma desempenhando diferentes ações. Para tal, decidimos que cada entidade teria na sua implementação duas threads: uma de comunicação e outra de trabalho. A primeira serve para suportar a criação do anel, a descoberta de entidades e a comunicação dentro do anel, já explicadas anteriormente. A outra serve para realizar as tarefas de simulação inerentes a cada entidade, que serão explicadas mais à frente. Por isso, ao criar um objeto de cada uma das diferentes classes, inicia-se imediatamente a thread de trabalho e inclui-se, por composição, um objeto da classe ringNode em cada uma, iniciando assim a thread de comunicação. Para trocar informação entre as duas, criámos duas filas (queues) na classe ringNode: uma para receber pedidos da thread de trabalho e outra para enviar pedidos para a mesma thread. Portanto, após o processo de descoberta das entidades, a thread de comunicação irá estar à espera de receber pedidos do seu predecessor durante um período de tempo (timeout). Se receber alguma mensagem, verifica se é para a sua entidade ou não. Em caso afirmativo, coloca essa mensagem na fila de pedidos a enviar para a thread de trabalho. Caso contrário, reencaminha a mensagem para o seu sucessor. No entanto, caso não receba nenhuma mensagem do seu predecessor, a thread de comunicação verifica se existem pedidos na fila de pedidos recebidos por parte da thread de trabalho. Se existirem, ele envia esse mesmo pedido para o seu sucessor, contendo o identificador da entidade destino.

O comportamento da thread de trabalho é semelhante ao da thread de comunicação. Primeiro, fica à espera de receber pedidos por parte de um determinado Cliente. Caso não receba nenhum pedido durante um determinado intervalo de tempo (timeout), verifica se há pedidos na fila de pedidos enviados pela thread de comunicação. Em caso afirmativo, realiza uma determinada ação consoante o papel da sua entidade na simulação, e envia uma mensagem, se necessário, para a fila de pedidos da thread de comunicação.

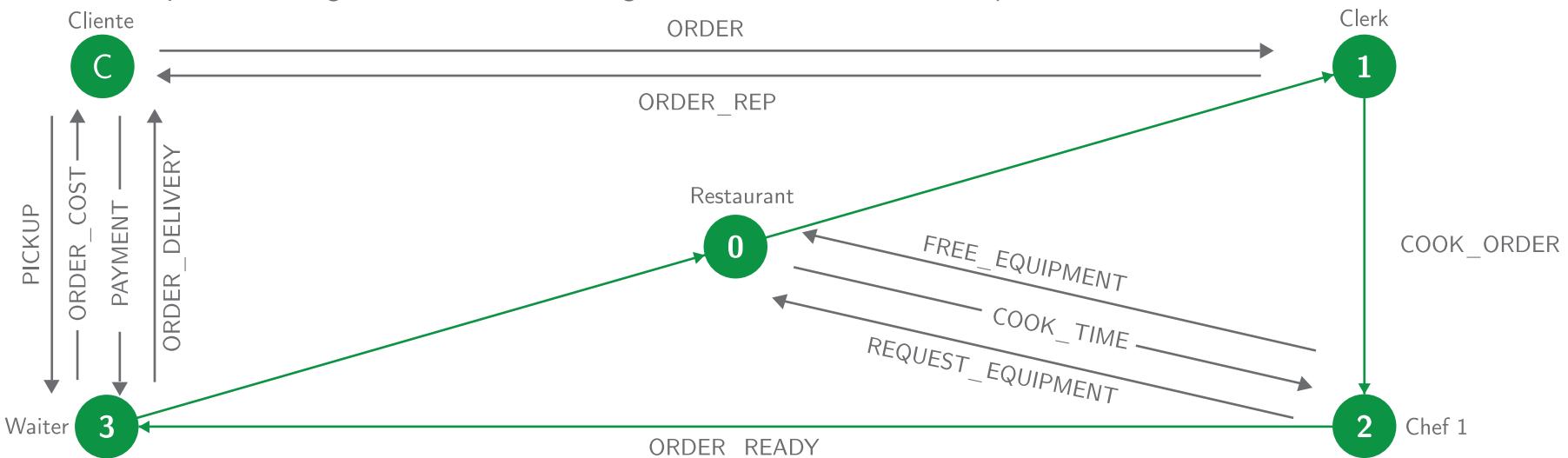
Este processo é cíclico.

3.1 Implementação de Simulação

Após a simulação ser iniciada, cada entidade fica à espera de receber um pedido por parte de um Cliente exterior ao anel, cada uma num determinado porto. A primeira mensagem que o Cliente envia ao anel é com um determinado pedido aleatório de comida, com um mínimo de 1 item e um máximo de 5 items. O responsável por anotar o pedido é o Recepcionista (Clerk). Por isso, se alguma das restantes entidades receber um pedido por parte do Cliente (método 'ORDER'), envia-o pelo anel até chegar ao Recepcionista, juntamente com o endereço do Cliente. O Recepcionista anota o pedido, atribui-lhe um número aleatório gerado com a libraria `uuid`, e envia ao Cliente o número do pedido. O Cliente recebe o número do pedido e envia à entidade com que iniciou a comunicação uma mensagem para o levantar (método 'PICKUP'). Esta mensagem é reencaminhada pelo anel até chegar ao Empregado de Balcão, que guarda o pedido e o endereço do Cliente num dicionário de pedidos pendentes. Após enviar o número do pedido ao Cliente, o Recepcionista envia o pedido recebido para o Cozinheiro, para que este o comece a preparar. Antes de o fazer, envia uma mensagem à entidade do Restaurante para requisitar os equipamentos da cozinha, conforme o número de items do pedido. O Restaurante, ao receber esta mensagem, calcula um tempo de preparação aleatório, segundo uma distribuição gaussiana específica para cada um dos equipamentos requisitados, e retorna este valor de tempo ao Cozinheiro, novamente através de uma mensagem no anel. O Cozinheiro, ao receber este tempo de preparação, simula que cozinha os items (através de um método `sleep`), interrompendo a thread de trabalho. Após ter acabado de cozinhar, envia uma mensagem ao Restaurante a dizer que já pode libertar os equipamentos da cozinha. Quando o pedido estiver totalmente pronto, o Cozinheiro envia-o ao Empregado de Balcão. O Empregado de Balcão, ao receber o pedido confeccionado, calcula o preço total do mesmo e envia o preço ao Cliente. O Cliente recebe a quantia a pagar, efetua o pagamento ao Empregado de Balcão e aguarda pela entrega. Após receber o pagamento, o Empregado de Balcão envia o pedido pronto ao Cliente, e este último abandona o Restaurante.

No entanto, para melhorarmos a performance da nossa solução, decidimos introduzir mais um Cozinheiro. Para decidirmos qual dos dois Cozinheiros deveria ser responsável por confeccionar um determinado pedido, utilizámos o algoritmo de Round Robin, atribuindo os pedidos novos alternadamente pelos dois Cozinheiros.

Em baixo segue-se um exemplo de um diagrama da troca de mensagens no anel referentes à simulação.



Restaurante Drive-Through Peer-to-Peer

4.1 Teste da solução

Para verificar os resultados da nossa solução, todas as trocas de mensagens e ações realizadas são demonstradas no terminal através da utilização de logs, como se exemplifica na figura em baixo. Estes logs contêm informação sobre cada entidade, bem como se são referentes à thread de comunicação (COMM) ou à thread de simulação (SIMU).

Para executar a solução, é necessário correr um dos seguintes scripts em bash:

- run.sh, que em que o endereço de acesso ao anel é o do Restaurante
 - run_random.sh, em que escolhe um endereço aleatório de uma determinada entidade para ser o endereço de acesso ao anel

É de salientar que ambos os scripts iniciam múltiplos Clientes, cada um num determinado porto.

