Drive-Through Restaurant P2P

Trabalho realizado por:

- Pedro Escaleira - 88821
- Rafael Simões - 88984

Implementação do Ring

Como foi pedido, implementamos cada entidade (excepto o cliente) como sendo um nó pertencente a uma estrutura em anel. Contudo, decidimos ir um pouco mais além do que era pretendido, e ao invés de a entidade **Restaurante** ter de ser a primeira entidade no ***Ring*** e as restantes terem de a ela se ligar, implementamos uma solução onde as entidades se podem ligar por qualquer ordem, sendo que passado algum tempo se ligam de acordo com a referida estrutura e na ordem pedida nos requisitos do trabalho. Desta forma, na nossa aproximação, todos os nós são iguais uns aos outros, sendo que se podem especializar de acordo com a tarefa que lhes é destinada (o **Chef** tem de cozinhar os pedidos dos clientes, o **Restaurante** tem de possibilitar ao **Chef** cozinhar, utilizando os seus equipamentos, etc.).

Para que o sistema acima referido funcionasse, foi necessário implementar também um sistema que elegesse o nó coordenador, já que no nosso caso não é elegante assumir que é o **Restaurante**, nó este que é responsável por pedir a cada nó que envie para a rede o seu *id* de acordo com a entidade que representa (**Restaurante** -> *id*=0, **Rececionista** -> *id=1*, **Chef** -> *id*=2, **Empregado** -> *id*=3) e, quando o nó está formado e cada um sabe quem é quem, enviar o ***TOKEN*** para a rede.

É também importante referir que antes do ***TOKEN*** ser enviado para a rede, todas as mensagens trocadas entre as entidades são enviadas em broadcast (através dum ciclo *for*) para a rede 127.0.0.0/21, na porta 5000.

Mensagens na rede

Para as mensagens enviadas na rede, baseamo-nos numa aproximação parecida com o que acontece com os pacotes que circulam em "redes normais", onde cada protocolo corresponde a um tipo de mensagem diferente, com diferentes *keys* e onde cada mensagem pode encapsular outras mensagens de outros protocolos. Desta forma, temos 7 tipo de mensagens (ou protocolos) diferentes para enviar informação. Na rede entre os nós e entre os nós e os clientes, circulam sempre mensagens do tipo {'method': *X*, 'args': *X*'}. Qualquer outro tipo de mensagem é sempre encapsulada dentro desta ao ser enviada para a rede.

Para que este encapsulamento seja feito de forma mais facilitada, criamos uma função para cada um deles num documento à parte, que o código de cada entidade pode importar e usar, obtendo a mensagem já estruturada, sem necessidade de haver "poluição de código".

Implementação da simulação

- **Rececionista**
  - Recebe o pedido de comida do cliente, enviando o respetivo pedido para o **Chef** e atribui um número de pedido ao cliente.
- **Chef**
  - Recebe o pedido enviado pelo **Rececionista** e cozinha os respetivos alimentos , fazendo requests ao restaurante para uso dos equipamentos , devolvendo-os no fim do processo de confecção de cada alimento estar terminado.
  - No fim do pedido estar totalmente cozinhado este envia envia este para o **Empregado**.
- **Empregado**
  - Além de receber os pedido de comida confeccionados pelo **Chef**, também recebe as ordens de pedido de comida, significando que o cliente está pronto para receber o pedido.
  - Quando recebe um pedido de comida, esta verifica se o seu número corresponde ao número que tinha sido atribuído a esse cliente, e entrega-lhe a comida caso isso se verifique.
  - Esta entidade também tem uma variável (temporal) de controlo da qualidade da comida, que caso este valor seja excedido, a comida confecionada deixa de ser comestível (devido ao facto de a comida ficar fria e esta não comprimir os padrões de qualidade do restaurante) e consequentemente a comida não será entregue (evitando que este pedido fique eternamente na lista de espera).
- Restaurante
  - Recebe os pedidos de equipamentos por parte do **Chef**, e caso não estejam a ser usados, os envia para ele. Também recebe novamente os equipamentos que tinham sido pedidos.

Segunda versão

Para além do que era pedido, e para além de termos implementado a entrada dos nós na rede de forma mais elegante do que era suposto, decidimos também implementar uma segunda versão do trabalho, com o intuito de tornar a rede em anel ainda mais robusta. Basicamente, nesta segunda versão, projetamos o ***Ring*** de forma a que os nós possam entrar e sair da rede sem causar nenhum problema. Isto claro só funciona caso nenhum cliente esteja a ser atendido, já que nesse caso, o ***TOKEN*** deixa de circular na rede até que o **Anel** esteja novamente formado.

Para isso ser possível, foi necessário implementar o envio de mensagens com um *time stamp*, que são enviadas para verificar se o sucessor de cada nó continua ativo. Caso o sucessor não responda dentro de um certo intervalo de tempo, o predecessor pensa que o sucessor "morreu" e informa toda a rede do sucedido, sendo que o ***TOKEN*** deixa de ser enviado e os nós ficam à espera de que a referida entidade se ligue novamente. Quando isso acontece, tudo ocorre naturalmente, como acontece quando um nó entra de "forma normal" no **Anel**.

Outros pontos importantes

- Para termos mais certezas de que a rede projetada não tinha problemas, implementamos dois *scripts*, um para cada versão, que executam cada uma delas para um número alargado de clientes (lançados ao mesmo tempo) e que guarda os dados dessa execução num ficheiro de texto, para no caso de haver algum eventual erro, o podermos identificar mais facilmente. Para verificar se a segunda versão funcionava de acordo com o que era suposto, o script mata e volta a pôr em execução entidades aleatórias em ordem aleatória, lançando de seguida clientes.
- Foi também feito um script para mudar de forma mais facilitada a versão do ***Ring*** que as entidades usam.
- Pelo facto de no início não estarmos cientes de que era obrigatório usar a versão do cliente cedida pelos professores, fizemos todo o trabalho com uma versão criada por nós. Desta forma, quando o soubemos, para não "remendar" o código e evitar possíveis erros dessa ação, criamos um **Adaptador** que é chamado para qualquer mensagem recebida pelos nós, sendo que se for uma do cliente dos professores, a mensagem é convertida de acordo com as especificações que já tínhamos. Desta forma, tanto o cliente por nós criado, como o dos professores, funciona.