Restaurante

EXECUÇÃO E SINCRONIZAÇÃO DE PROCESSOS E THREADS

Abel José Enes Teixeira 113655| Diogo Lopes Oliveira 113664

31/12/2023

Sistemas Operativos

Prof. António Guilherme Rocha Campos

Prof. Regente José Nuno Panelas Nunes Lau

Ano letivo

2023/2024

deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

ÍNDICE

Introdução

O projeto em questão visa a aprendizagem no que toca à execução e sincronização de processos e threads.

O exercício passa por simular um restaurante que implementa 4 atores que irão desempenhar diferentes funções, que terão influência sobre as restantes entidades.

1. **Chef** – recebe os pedidos do empregado, prepara a comida e de seguida solicita ao empregado para levar o pedido à mesa.

2. **Empregado de Mesa** – recolhe os pedidos dos grupos e faz a ligação entra a cozinha e os grupos, entregando os pedidos.

3. **Grupos** – dirigem-se ao rececionista que lhes atribui a mesa, de seguida pedem comida ao empregado, esperam pelo pedido, pagando a conta no fim da refeição.

4. **Rececionista** – indica a mesa ao grupo ou solicita que espere caso as mesas estejam ocupadas. No fim recebe os pagamentos.

Análise superficial da execução do código

Com o diagrama de sequências abaixo incluído, podemos ter uma melhor perceção do decorrer de todas as atividades e o papel de cada ator responsável por cada uma dessas atividades.

Aqui está descrito todo o processo que envolve o pedido de um grupo.

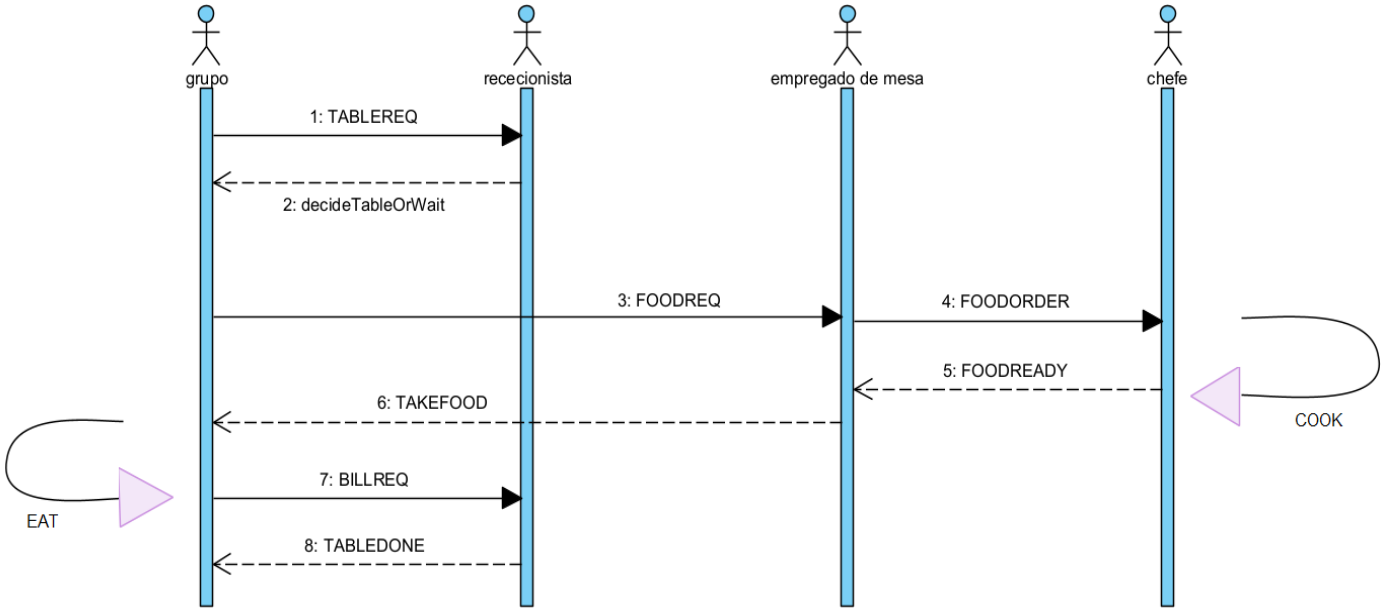

Figura 1 - Diagrama de Sequências sobre o trabalho

Análise superficial da execução do código

Para facilitar a compreensão e a organização do código, optámos por dispor de todos os Up's e Down's a efetuar para a concretização de um pedido com sucesso.

1) **receptionistReq**
   a) **UP**
      i) **Grupo** - Quando se dirigem ao rececionista, tanto para pedir mesa, como no momento do pagamento
   b) **DOWN**
      i) **Rececionista** – Quando começa a espera por um grupo que se diriga para pedir mesa ou pagar

2) **receptionistRequestPossible**
   a) **UP**
      i) **Rececionista** – Após atribuir uma mesa a um grupo ou metê-los em espera
   b) **DOWN**
      i) **Grupo** – Antes de pedir mesa ou a conta

3) **waiterRequest**
   a) **UP**
      i) **Grupo** – Quando o empregado recolhe o pedido
      ii) **Chefe** – Quando acaba de processar o pedido e notifica o empregado para o entregar

   b) **DOWN**

i) **Empregado de mesa** – Quando espera por um pedido da mesa ou pela notificação do chef

4) **waitOrder**
   a) **UP**
      i) **Empregado de mesa** – Quando recolhe o pedido de um grupo ou recebe notificação do chef
   b) **DOWN**
      i) **Chefe** – Quando entra em rest e espera pelo pedido do próximo grupo

5) **orderReceived**
   a) **UP**
      i) **Chefe** – após receber o pedido de um grupo
   b) **DOWN**
      i) **Empregado de mesa** – após entregar um pedido ao chefe

6) **waitForTable[id]**
   a) **UP**
      i) **Rececionista** – quando acaba de atribuir uma mesa a um grupo com um determinado ID
   b) **DOWN**
      i) **Grupo** – no momento em que efetua o pedido para atribuição de mesa

Análise superficial da execução do código

7) **requestReceived[table]**
   a) **UP**
      i) **Empregado de Mesa** – quando recebe o pedido de uma mesa (table) de um grupo (id)
   b) **DOWN**
      i) **Grupo** – quando acaba o pedido que será levado pelo empregado de mesa

8) **foodArrived[table]**
   a) **UP**
      i) **Empregado de Mesa** – quando entrega a comida a um grupo numa determinada mesa
   b) **DOWN**
      i) **Grupo** – no momento em que pede a conta ao rececionista

Chefe

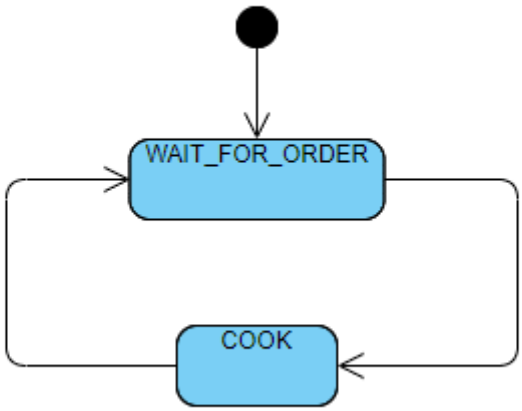

Figura 2 - Diagrama de Estados do Chefe

waitForOrder()

Esta função desempenha um papel crucial ao manter o chefe à espera de um novo pedido do empregado de mesa.

Inicialmente, é realizado um "Down" no semáforo waitOrder.

Dentro da região crítica, o chefe identifica o grupo responsável pelo pedido, procedendo em seguida à alteração do seu estado para COOK. De seguida, efetua um "Up" no semáforo orderReceived, comunicando ao empregado de mesa que o pedido foi recebido.

processOrder()

Função na qual ocorre o processo de preparação e entrega da comida por parte do chefe.

É usado um usleep para representar o tempo de confeção do pedido e de seguida faz-se um Down no semáforo waiterRequestPossible para solicitar o empregado de mesa, sendo o estado do chef alterado de seguida para WAIT_FOR_ORDER

Na zona crítica, o empregado de mesa é avisado que o pedido está pronto para entrega.

É realizado no fim um UP no waiterRequest na eventualidade de surgir um novo pedido.

Empregado de Mesa

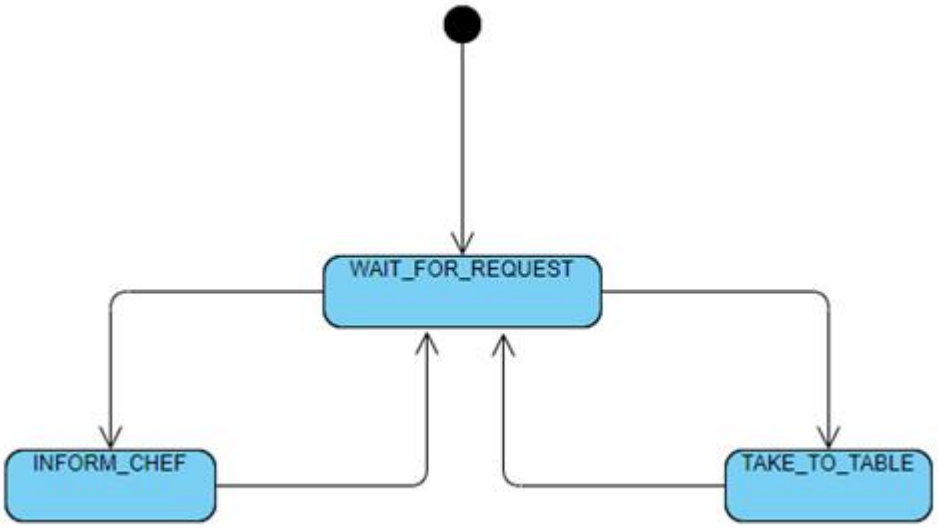

Figura 3 - Diagrama de estados do empregado de mesa

waitForClientOrChef()

Conforme o próprio nome sugere, esta função tem a responsabilidade de manter o empregado a aguardar um pedido, seja de um grupo ou do chefe. Inicialmente, o acesso à zona crítica é feito para alterar o estado do empregado para WAITING_ORDER. Em seguida, fora da zona crítica, é realizado um DOWN no semáforo waiterRequest, mantendo o empregado à espera de um pedido.

Assim que um pedido é recebido, uma nova entrada na região crítica é efetuada para registar os detalhes do pedido, com a identificação do grupo ao qual se refere e seu tipo.

Esses dados são posteriormente retornados pela função. Por fim, é realizado um UP no waiterRequestPossible, indicando que o empregado de mesa está novamente disponível para ser solicitado.

informChef()

Esta função é executada caso o pedido seja feito por um grupo. Inicialmente, o acesso à zona crítica é realizado, onde o estado do empregado é alterado para INFORM_CHEF, e o pedido é feito ao chefe (com um UP no semáforo waitOrder) para que ele prepare a refeição para o grupo identificado na função waitForClientOrChef.

Por último, é feito um DOWN no orderReceived, permitindo que o empregado de mesa aguarde a confirmação de que o chefe recebeu o pedido. Em seguida, é realizado um UP no requestReceived do respetivo grupo, informando-o de que o pedido foi recebido e que o chefe já foi notificado.

takeFoodToTable()

Esta função é executada quando o pedido é feito pelo chefe, indicando que a refeição está pronta para ser servida. Dentro da zona crítica, o estado do empregado de mesa é alterado para TAKE_TO_TABLE, e um UP no semáforo foodArrived do grupo correspondente é realizado para informar que a comida está pronta para ser servida.

Grupo

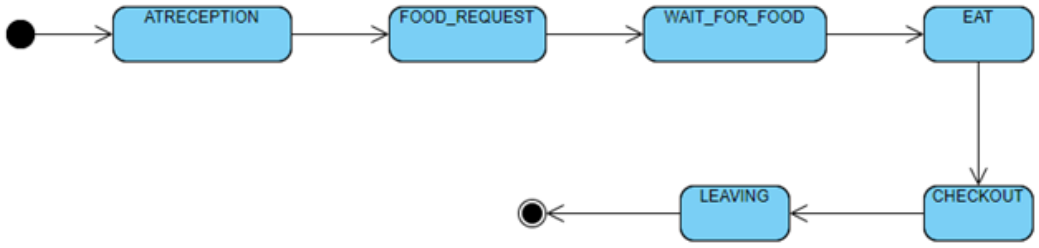

Figura 4 - Diagrama de estados do grupo

checkInAtReception()

Função em que ocorre o pedido de mesa por parte do grupo.

Primeiramente, antes da região crítica, é feito um DOWN no receptionistRequestPossible para solicitar o rececionista.

Solicitado o rececionista, o estado do grupo é mudado para ATRECEPTION e é feito um DOWN ao semáforo waitForTable

orderFood()

Nesta função o grupo faz o pedido ao empregado de mesa.

Inicialmente, solicita-se o empregado com um DOWN no semáforo waiterRequestPossible e na zona critica, mudamos o estado para FOOD_REQUEST.

Para efetuar o pedido, damos um DOWN para requestReceived.

waitFood()

Na zona crítica, o estado do grupo é alterado para WAIT_FOR_FOOD.

De seguida é feito um DOWN em foodArrived da mesa de cada grupo e por último, quando o pedido chega à mesa, o estado do grupo é atualizado para EAT

checkOutAtReception()

Terminada a refeição, o grupo pede a conta, solicitando o rececionista com um DOWN receptionistRequestPossible.

Mal o empregado tenha disponibilidade, o estado do grupo é alterado para checkout e o grupo faz o pedido da conta através de BILLREQ.

O rececionista é notificado com um UP através de receptionistReq. Ao mesmo tempo, tableDone recebe DOWN da mesa onde estava sentado o grupo já em checkout. Terminado o pagamento, o estado do grupo é alterado para LEAVING.

Rececionista

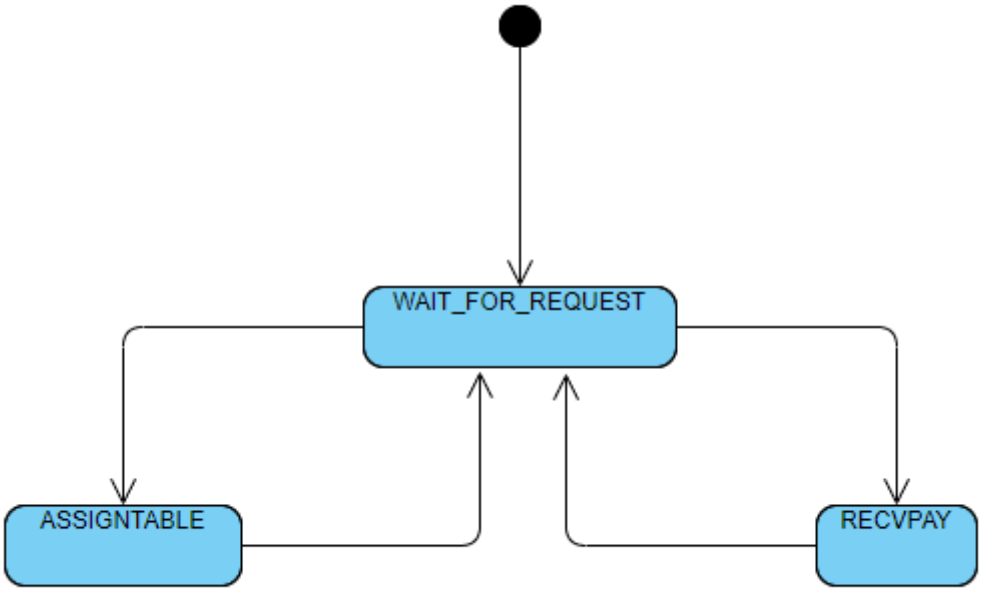

Figura 5 - Diagrama de estados do rececionista

waitForGroup()

Esta função coloca o rececionista à espera de um novo pedido vindo de um grupo. Inicialmente, ocorre um acesso à região crítica para alterar o estado do mesmo para WAITING_ORDER. Posteriormente, fora da zona crítica, é realizado um DOWN no semáforo recepcionistReq, efetivamente colocando o rececionista à espera de um grupo.

Quando o rececionista é solicitado por um grupo, volta-se a aceder à zona crítica para registar os dados do pedido, que serão posteriormente retornados pela função. Antes de sair da região crítica, é realizado um UP no semáforo

recepcionistRequestPossible, assinalando que o rececionista pode ser novamente solicitado por um novo grupo.

provideTableOrWaitingRoom()

Se o pedido feito pelo grupo for do tipo TABLEREQ, ou seja, um pedido de mesa, esta é a função chamada. Primeiramente, o estado do rececionista é alterado para ASSIGNTABLE dentro da região crítica, indicando que está a tomar a decisão de fornecer ou não uma mesa ao grupo, ou deixá-lo em espera (WAIT).

A decisão é tomada por outra função, decideTableOrWait, que utiliza um algoritmo simples para verificar se existem mesas disponíveis. Se existir uma mesa para grupo, é realizado um UP no semáforo waitForTable do grupo correspondente; caso contrário, o grupo é mantido em espera.

receivePayment()

Esta função é chamada no caso de o pedido feito pelo grupo ser do tipo BILLREQ, ou seja, um pedido para solicitar a conta. Após entrar na zona crítica, o estado do rececionista é atualizado para RECVPAY, indicando que está a receber o pagamento da refeição. Em seguida, é realizado um UP no semáforo tableDone da mesa onde o grupo estava,

informando-o de que o pagamento foi recebido com sucesso e que pode ir embora.

No entanto, é necessário verificar se a mesa desocupada pode ser novamente ocupada por outros grupos que estejam à espera. Para isso, o estado do rececionista é novamente alterado para ASSIGNTABLE e é utilizada a função decideNextGroup, decidindo qual o próximo grupo, caso seja possível, a ocupar a mesa agora disponível. Se for possível, é realizado um UP no semáforo waitForTable associado ao grupo, permitindo-lhe dirigir-se à mesa.

Testes

Para avaliar o resultado, adotamos uma abordagem distinta da sugerida.

Em vez de testar progressivamente os valores obtidos, decidimos preencher os diversos ficheiros conforme o ciclo necessário para uma refeição e, somente ao final, verificar se os valores estavam em conformidade com as expectativas.

A partir desse ponto, realizamos as alterações necessárias. Em razão dessa abordagem, procedemos à modificação simultânea dos diferentes ficheiros semSharedMem.

Também foi crucial para a testagem o ficheiro run.sh que executa o código 1000 vezes seguidas.

Restaurant - Description of the internal state

CH	WT	RC	G00	G01	G02	G03	G04	gWT	T00	T01	T02	T03	T04
0	0	0	1	1	1	1	1	0	.	.	.	.	.
0	0	0	1	1	1	1	1	0	.	.	.	.	.
0	0	0	1	1	1	1	1	0	.	.	.	.	.
0	0	0	1	2	1	1	1	0	.	.	.	.	.
0	0	1	1	2	1	1	1	0	.	.	.	.	.
0	0	1	1	2	2	1	1	0	.	0	.	.	.
0	0	0	1	2	2	1	1	0	.	0	.	.	.
0	0	0	1	3	2	1	1	0	.	0	.	.	.
0	0	1	1	3	2	1	1	0	.	0	.	.	.
0	0	0	1	3	2	1	1	0	.	0	1	.	.
0	1	0	1	3	2	1	1	0	.	0	1	.	.
0	1	0	1	3	3	1	1	0	.	0	1	.	.
1	1	0	1	3	3	1	1	0	.	0	1	.	.
1	0	0	1	3	3	1	1	0	.	0	1	.	.
1	0	0	1	4	3	1	1	0	.	0	1	.	.
1	1	0	1	4	3	1	1	0	.	0	1	.	.
0	1	0	1	4	3	1	1	0	.	0	1	.	.
1	1	0	1	4	3	1	1	0	.	0	1	.	.
1	0	0	1	4	3	1	1	0	.	0	1	.	.
1	0	0	1	4	4	1	1	0	.	0	1	.	.
1	2	0	1	4	4	1	1	0	.	0	1	.	.
1	0	0	1	4	4	1	1	0	.	0	1	.	.
1	0	0	1	5	4	1	1	0	.	0	1	.	.
0	0	0	1	5	4	1	1	0	.	0	1	.	.
0	2	0	1	5	4	1	1	0	.	0	1	.	.
0	0	0	1	5	4	1	1	0	.	0	1	.	.
0	0	0	1	5	5	1	1	0	.	0	1	.	.
0	0	0	1	5	5	2	1	0	.	0	1	.	.
0	0	1	1	5	5	2	1	0	.	0	1	.	.
0	0	0	1	5	5	2	1	1	.	0	1	.	.
0	0	0	1	5	5	2	2	1	.	0	1	.	.
0	0	1	1	5	5	2	2	1	.	0	1	.	.
0	0	0	1	5	5	2	2	2	.	0	1	.	.
0	0	0	2	5	5	2	2	2	.	0	1	.	.
0	0	1	2	5	5	2	2	2	.	0	1	.	.
0	0	0	2	5	5	2	2	3	.	0	1	.	.
0	0	0	2	5	6	2	2	3	.	0	1	.	.
0	0	2	2	5	6	2	2	3	.	0	1	.	.
0	0	1	2	5	6	2	2	3	.	0	.	.	.
0	0	0	2	5	6	2	2	2	1	0	.	.	.
0	0	0	2	5	7	2	2	2	1	0	.	.	.
0	0	0	3	5	7	2	2	2	1	0	.	.	.
0	1	0	3	5	7	2	2	2	1	0	.	.	.
1	1	0	3	5	7	2	2	2	1	0	.	.	.
1	0	0	3	5	7	2	2	2	1	0	.	.	.
1	0	0	4	5	7	2	2	2	1	0	.	.	.
0	0	0	4	5	7	2	2	2	1	0	.	.	.
0	2	0	4	5	7	2	2	2	1	0	.	.	.
0	0	0	4	5	7	2	2	2	1	0	.	.	.
0	0	0	5	5	7	2	2	2	1	0	.	.	.
0	0	0	6	5	7	2	2	2	1	0	.	.	.
0	0	2	6	5	7	2	2	2	1	0	.	.	.
0	0	1	6	5	7	2	2	2	.	0	.	.	.

```
0  1  0    7  5  7  3  2  1  .  0  .  1  .
1  1  0    7  5  7  3  2  1  .  0  .  1  .
1  0  0    7  5  7  3  2  1  .  0  .  1  .
1  0  0    7  5  7  4  2  1  .  0  .  1  .
0  0  0    7  5  7  4  2  1  .  0  .  1  .
0  2  0    7  5  7  4  2  1  .  0  .  1  .
0  0  0    7  5  7  4  2  1  .  0  .  1  .
0  0  0    7  5  7  5  2  1  .  0  .  1  .
0  0  0    7  5  7  6  2  1  .  0  .  1  .
0  0  2    7  5  7  6  2  1  .  0  .  1  .
0  0  1    7  5  7  6  2  1  .  0  .  .  .
0  0  0    7  5  7  6  2  0  .  0  .  .  1
0  0  0    7  5  7  7  2  0  .  0  .  .  1
0  0  0    7  5  7  7  3  0  .  0  .  .  1
0  1  0    7  5  7  7  3  0  .  0  .  .  1
1  1  0    7  5  7  7  3  0  .  0  .  .  1
1  0  0    7  5  7  7  3  0  .  0  .  .  1
1  0  0    7  5  7  7  4  0  .  0  .  .  1
0  0  0    7  5  7  7  4  0  .  0  .  .  1
0  2  0    7  5  7  7  4  0  .  0  .  .  1
0  2  0    7  5  7  7  5  0  .  0  .  .  1
0  2  0    7  5  7  7  6  0  .  0  .  .  1
0  2  2    7  5  7  7  6  0  .  0  .  .  1
0  2  0    7  5  7  7  6  0  .  0  .  .  .
0  2  0    7  5  7  7  7  0  .  0  .  .  .
0  2  0    7  6  7  7  7  0  .  0  .  .  .
0  2  2    7  6  7  7  7  0  .  0  .  .  .
0  2  2    7  7  7  7  7  0  .  .  .  .  .
```

Figura 6 - Output do programa

Conclusão

Após investir um considerável período tempo na deteção de erros, como por exemplo, o facto de o chefe começar a processar o pedido antes de sequer o receber.

O desfecho do projeto revela de maneira evidente os efeitos positivos alcançados, nomeadamente uma maior competência no que toca a lidar com execução e sincronização de processos e threads.

Este projeto para além do contributo para o conhecimento prático, teve também impacto na componente mais teórica, na medida em que uma componente implica o conhecimento sobre a outra.