cONTROLO Do FLUXO de um restaurante usando semÁforos

Sistemas Operativos

Tomás Costa - 89016

Índice

[Introdução 2](#_Toc534512532)

[Antes de Implementar 3](#_Toc534512533)

[Análise de Interações 4](#_Toc534512534)

[Constantes, Semáforos e Estados 5](#_Toc534512535)

[Entidade Chef 6](#_Toc534512536)

[Entidade Waiter 7](#_Toc534512537)

[Entidade Receptionist 8](#_Toc534512538)

[Entidade Grupo 9](#_Toc534512539)

[Problemas Encontrados 10](#_Toc534512540)

[Resultados Obtidos 11](#_Toc534512541)

[Conclusões 12](#_Toc534512542)

[Bibliografia 13](#_Toc534512543)

# Introdução

Como segundo trabalho prático da unidade curricular de Sistemas Operativos, foi pedido aos alunos que desenvolvessem uma aplicação em linguagem C que simulasse o fluxo de um restaurante com o uso de semáforos.

Um dos principais objetivos deste problema proposto passa pela compreensão dos mecanismos associados à execução e sincronização de processos e threads.

Este projeto tem como base código C fornecido pelo docente, pelo que será preciso apenas alterar funções nas 4 entidades que interagem no restaurante, sendo estas: o chefe, o garçom, o rececionista, e os grupos que chegam ao restaurante.

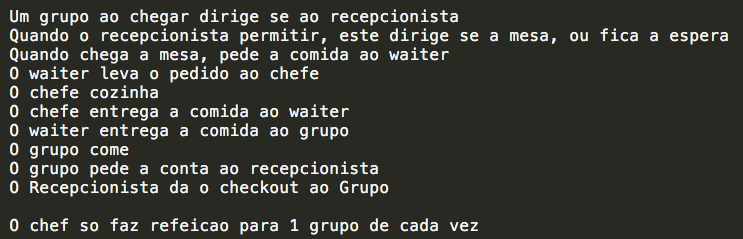
# Antes de Implementar

O pasta fornecida pelo docente contem 3 diretorias:

* Run – contem os executáveis
* Src – contem todo o código usado
* Doc – contem documentação usando o comando doxygen



Segundo o enunciado do trabalho estas são a interações que se realizam no ciclo de vida de um restaurante:



Caso queira testar as entidades do docente basta fazer $make all\_bin

Cada entidade pode ser testada individualmente, para isto basta fazer $make <entidade>

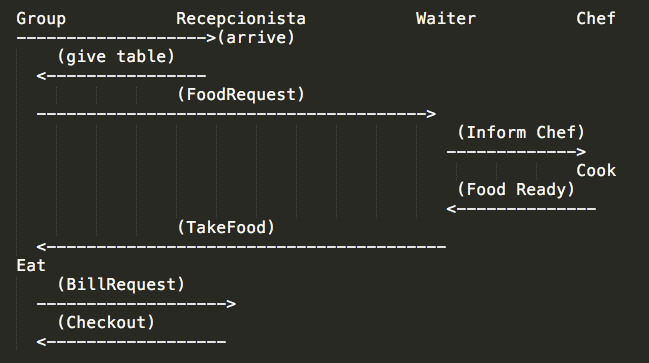
Para este trabalho é apenas necessário modificarmos 4 ficheiros denominados semSharedMem<entidade>.c, e dentro deste ficheiros alterar algumas funções que estão assinaladas com um ToDo.

É relevante ainda mencionar que posteriormente irei mencionar tanto um garçom como um waiter, embora esteja a referir a mesma entidade (há um cruzamento de linguagem).

# Análise de Interações

Antes de começar a escrever código, analisei muito bem o código fornecido, principalmente as constantes, os semáforos e a estruturas de dados partilhada.

Após esta análise, comecei a desenhar um quadro de interações para melhor entender que entidades interagiam com quais e qual era essa interação.

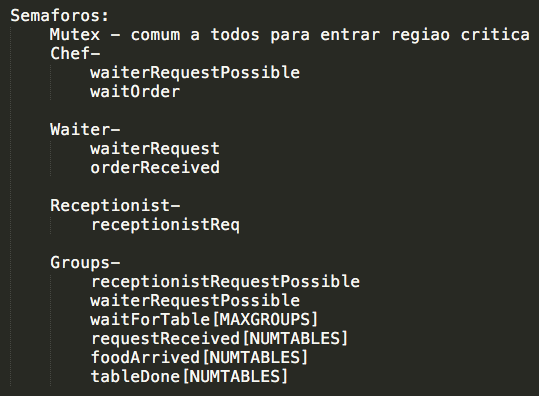


Nota: os estados não estão mencionados e o quadro tem algumas melhorias possíveis, mas foi uma primeira versão para melhor entender as interações

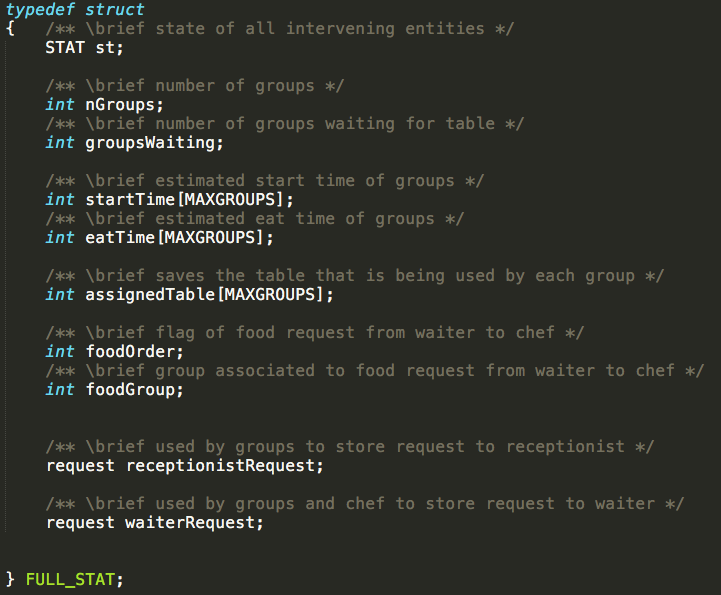
Após desenhar este quadro de interações e como sugerido pelo professor comecei pela entidade do chefe, pois e a que tem menos interações e menos funcionalidades.

# Constantes, Semáforos e Estados

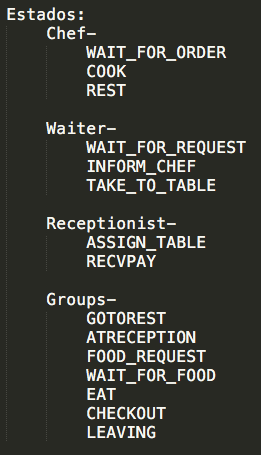
Foi bastante importante anotar todas as constantes de estados, a estrutura de dados chamada FULL\_STAT e STAT e também os semáforos que existiam no código fornecido, pois fui capaz de melhor perceber as interações que estavam a ocorrer e onde colocar certos semáforos

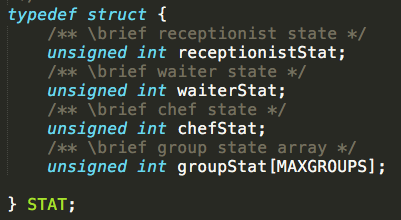


Nesta estrutura de dados estão presentes todas as flags, arrays e variáveis que iremos usar para as várias entidades:

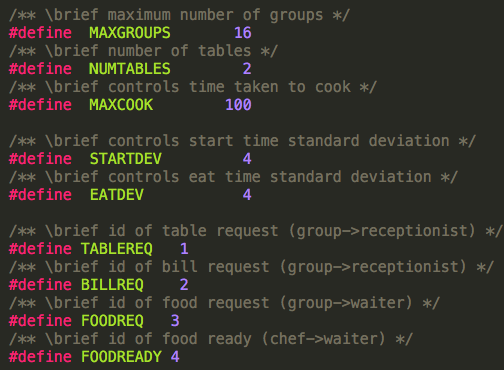


Esta imagem contem todos os estados que as diferentes entidades podem assumir:



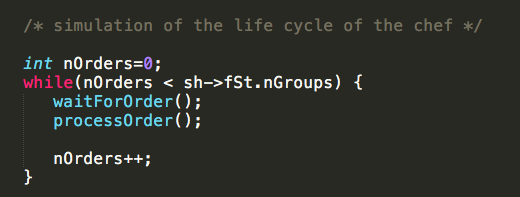


Importante também apontar estas constantes:



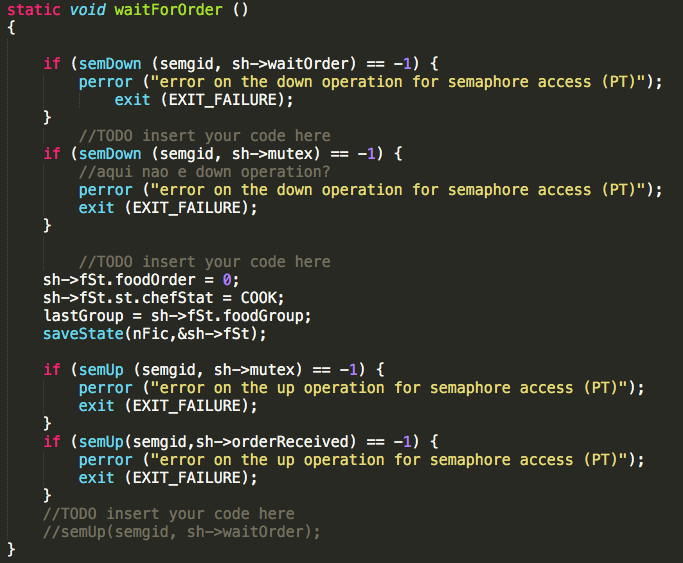
# Entidade Chef

O chefe interage apenas com o garçom e tem apenas duas funções (waitForOrder e processOrder) pelo que e uma entidade muito simples, como vemos aqui demonstrado pelo seu ciclo de vida:



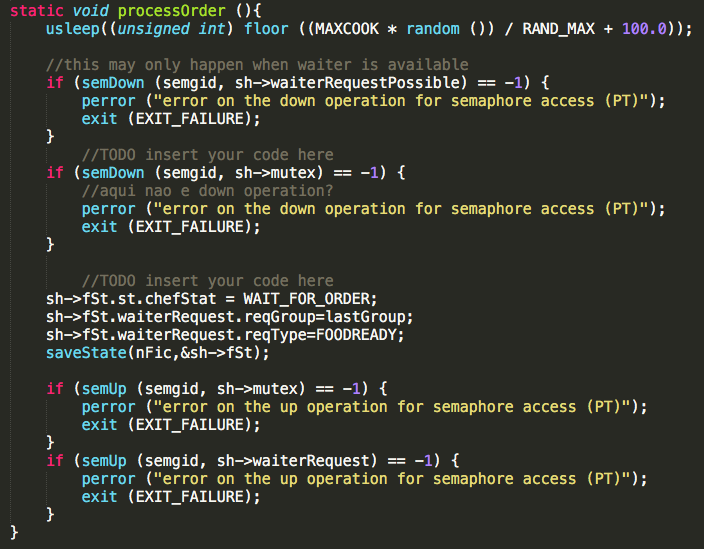
waitForOrder()

Para esta função e necessário o chef dar down ao semáforo waitOrder para poder esperar que chegue um pedido. Depois disso atribuímos valor de 0 a foodOrder pois e uma flag que server para mostrar que o chefe já esta a tratar de um pedido, alteramos o estado do chefe para COOK e atualizamos o valor de lastGroup para o valor em memória partilhada do grupo que pediu a comida e guardamos o estado, antes de terminarmos damos Up ao semáforo orderReceived, que o garçom deu Down para esperar que o chef cozinhe.



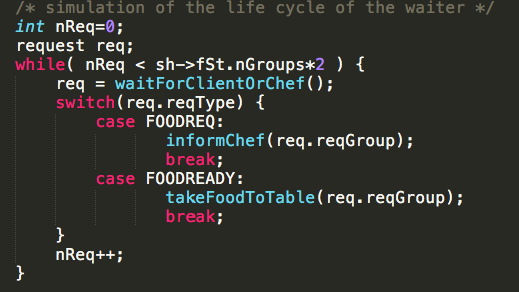
processOrder()

O chef gasta algum tempo para cozinhar, de seguida espera que o garçom fique disponível, depois disso atualizamos o estado do chef e atualizamos os valores do pedido com o número do ultimo grupo a pedir e com FOODREADY para sinalizar que a a comida esta pronta a levar, para terminar damos Up ao semáforo waiterRequest que o waiter da Down para esperar por pedidos.



# Entidade Waiter

O waiter interage com o chef e com os grupos e tem três funções, aqui está demonstrado o seu ciclo de vida.

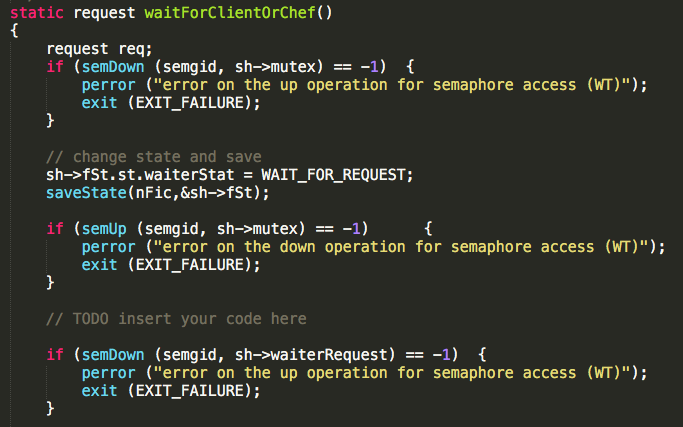


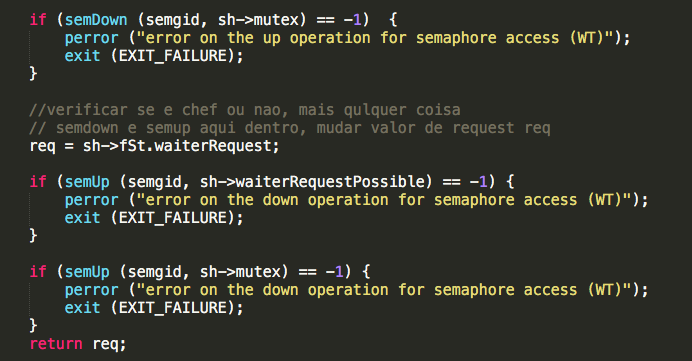
Daqui retiramos que a função waitForClientOrChef() funciona para ambos os pedidos e conforme o reqType muda que função vai ser executada posteriormente, pelo que temos que mudar o valor de req.reqType dentro da primeira função.

waitForClientOrChef()

Primeiro começamos por guardar o estado do garçom como a espera de pedidos e guardamos o seu estado.

Seguidamente fazemos o Down ao semáforo waiterRequest para esperar por pedidos, após este semáforo atualizamos o valor do pedido req com o valor do pedido guardado na estrutura de dados partilhada fSt.waiterRequest. Para terminar fazemos o Up ao semáforo waiterRequestPossible pois o waiter já está disponível para ser requisitado.



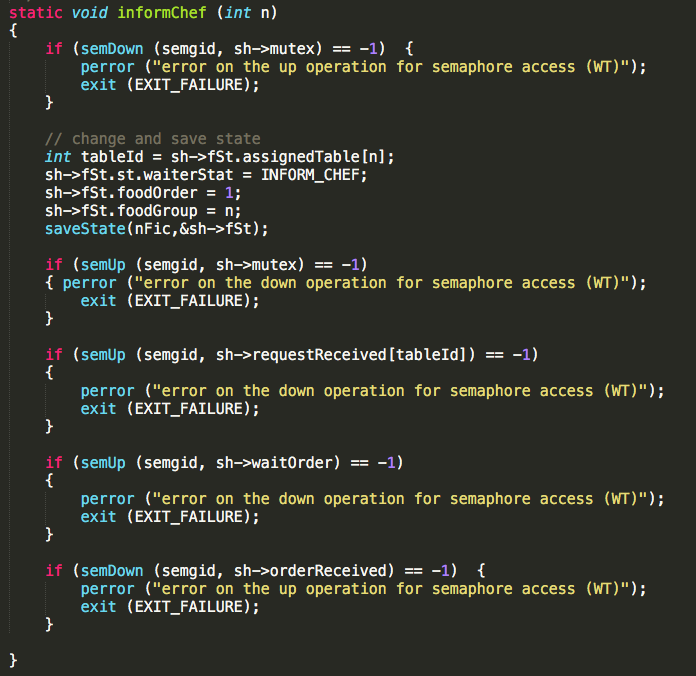


informChef()

Para informamos o chefe, começamos por mudar o estado do waiter, vamos buscar o ID da mesa com o array assignedTable do groupID passado como argumento, atualizamos a foodOrder para 1 para sinalizar que foi feito um pedido para o chefe e inserimos o valor do groupID(n) no foodGroup.

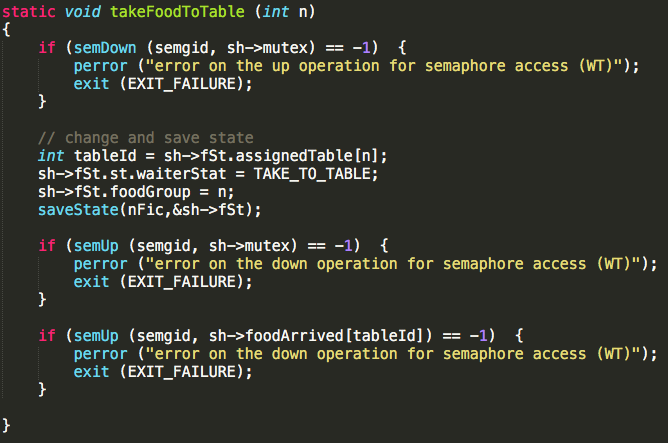
De seguida e feito o Up do requestReceived[tableID] pois e preciso sinalizar que o pedido dessa mesa foi feito e também Up a waitOrder para sinalizarmos o chef que estava a espera de pedidos.

Para finalizar e feito o Down a orderReceived para o waiter poder esperar que o chefe cozinhe.



takeFoodToTable()

A função takeFoodTable e bastante simples, basta mudarmos o estado do waiter, atualizarmos o foodGroup para o groupID passado como argumento, e no final dar Up ao semáforo foodArrived[tableID] (vamos buscar tableID da mesma forma que explicamos la em cima), para sinalizar chegada da comida a mesa, assim o grupo para de esperar.



# Entidade Receptionist

# Entidade Grupo

# Problemas Encontrados

Achei importante incluir esta secção pois penso que e devido à ocorrência de erros que nos mostra porque algumas soluções são melhores que outras.. say this better.

# Resultados Obtidos

Aa, Ter imagens com os resultados obtidos

# Conclusões

Concluindo o trabalho, é importante mencionar que o código não foi desenvolvido com o intuito de obter uma melhor otimização do mesmo, mas sim com o principal objetivo do funcionamento de todas as funcionalidades requeridas.

No entanto, muitas das funções desenvolvidas foram feitas de modo a adaptarem a mudanças de constantes definidas em probConst.c .

É também importante salientar que este trabalho e importante nesta altura pois é uma boa revisão dos mecanismos associados à execução e sincronização de processos e threads .

# Bibliografia

Slides da Unidade Curricular de Sistemas Operativos

https://www.geeksforgeeks.org/operating-system-dining-philosopher-problem-using-semaphores/

Documentação fornecida por Doxyfile