

**Sistemas Operativos**

**Trabalho Prático 2**

**Jantar de Amigos (Restaurant)**

**Professor:**

Nuno Lau ([*nunolau@ua.pt*](mailto:nunolau@ua.pt))

Realizado por:

Diogo Falcão, 108712, P3 – ??%

José Gameiro, 108840, P3 – ??%

02/01/2023

Índice

[***1. Introdução*** 3](#_Toc123489121)

[***2. Desenvolvimento*** 4](#_Toc123489122)

[**2.1. Material Fornecido** 4](#_Toc123489123)

# ***Introdução***

No âmbito da Unidade Curricular de Sistemas Operativos, foi-nos proposto realizar um trabalho prático, que consiste em simular um jantar de amigos, num restaurante, envolvendo três entidades, clients (clientes), waiter (empregado/a) e chef (chefe). Todas as entidades que participam nesta simulação são processos independentes, sendo que a sua sincronização e comunicação é efetuada através de vários semáforos e de memória partilhada.

A simulação é constituída por um grupo de vinte amigos, um empregado e um chefe. A simulação começa com a chegada de todos os amigos ao restaurante, em que o primeiro amigo a chegar será o que irá fazer o pedido da comida, que só o poderá fazer quando todos os amigos tiverem chegado, o último amigo a chegar será o que irá pagar a conta e só poderá pedi-la quando todos os amigos tiverem terminado a sua refeição. De modo a evitar situações em que 2 ou mais processos sejam bloqueados pois estão à espera de um evento que apenas pode ser despoletado por um dos processos em bloqueio, ou seja, uma situação de deadlock, iremos usar semáforos.

A utilização de semáforos serve essencialmente para controlar o acesso à memória partilhada, evitando choques entre as 3 entidades que participam na simulação. As notificações entre entidades são feitas através de semáforos para que o programa executasse sem encontrar problemas.

Com a realização deste trabalho prático, esperamos conseguir cumprir todos os pontos essenciais que são propostos no guião e alargar os nossos conhecimentos relativamente a programar com semáforos em C, visto ser um aspeto importante no que toca a controlar o acesso a determinadas regiões por parte de vários processos.

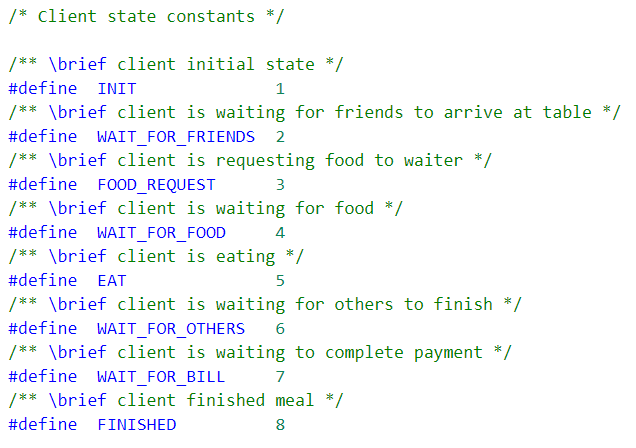
# ***Desenvolvimento***

## **Material Fornecido**

É disponibilizado um conjunto de ficheiros para a resolução do problema, em que está presente uma pasta src com vários ficheiros, estes contêm código em C para a simulação do jantar e, três deles, encontram-se incompletos que são semSharedMemChef.c, semSharedMemClient.c e semSharedMemWaiter.c e ao longo deste relatório iremos explicar como é que completámos as funções que se encontram incompletas em cada um dos ficheiros.

Existem também outros ficheiros que contém dados importantes para perceber como é que é simulado o jantar:

- probConst.h: este ficheiro contém variáveis que irão ser utilizadas depois nos 3 ficheiros incompletos, como os estados dos clientes, os estados do empregado, os estados do chefe, o tamanho máximo da mesa do restaurante e os tempos máximos para cozinhar a refeição e para comer.



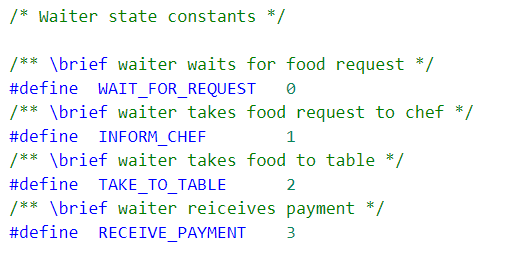
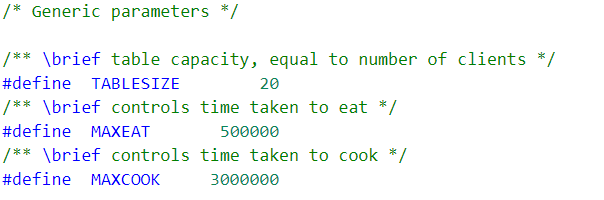
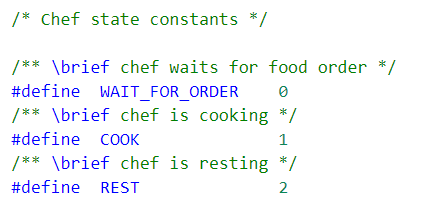
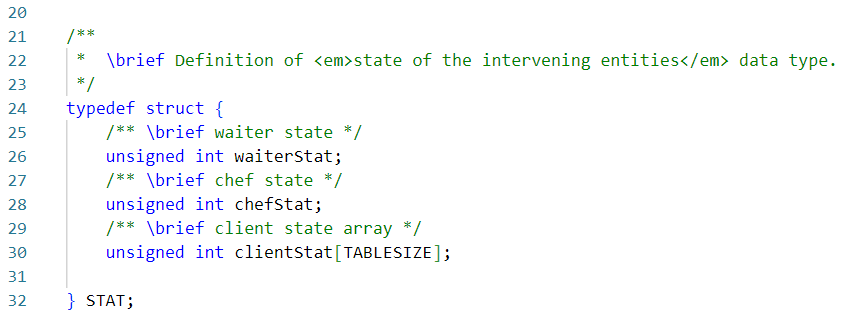


Fig.1 – Constantes definidas no ficheiro probConst.h.

- probDataStruct.h: neste ficheiro estão definidas duas estruturas que são, a estrutura STAT e a FULL\_STAT. A estrutura STAT tem como parâmetros 3 variáveis inteiras que são os estados de cada entidade envolvente no jantar, e como existem 20 clientes, logo a variável para o estado de cada cliente será um array com o tamanho de 20 (TABLESIZE). A estrutura FULL\_STAT apresenta vários parâmetros que encontram na figura dois, sendo que as variáveis nas linhas 48, 50, 52 e 54 irão funcionar com sinais, ou seja, o seu valor inicial é zero e quando for necessário estas variáveis serão postas a um; a variável tableFirst e tableLast irão guardar os ID’s do primeiro e último cliente a chegar, respetivamente; por fim as variáveis tableClients e tableFinishEat irão funcionar como contadores para que sempre que chegar um cliente ao restaurante ou um cliente acabar de comer estas variáveis são incrementadas.



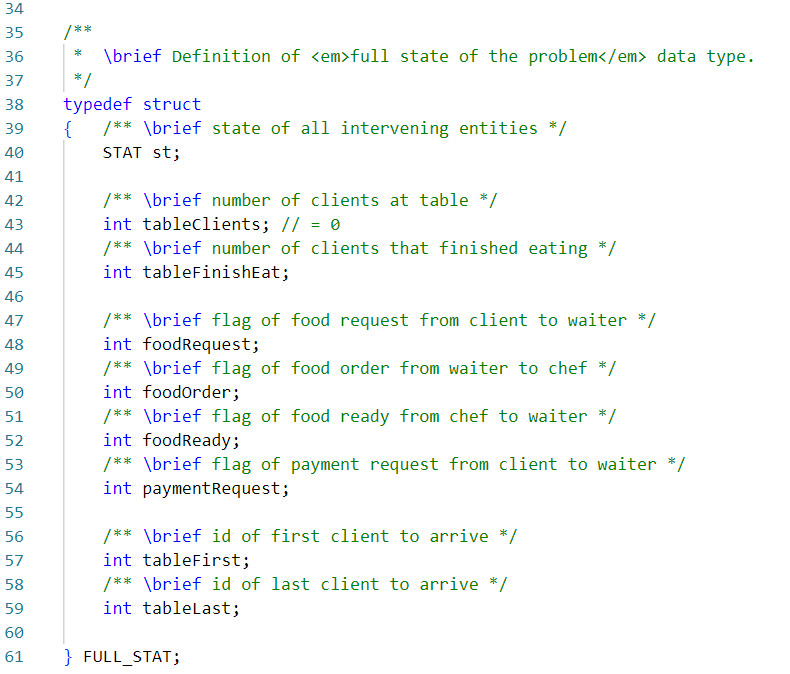


Fig.2 –Estruturas definidas no ficheiro probDataStruct.h.

- sharedDataSync.h: este ficheiro contém uma estrutura (SHARED\_DATA), que contém como parâmetros uma estrutura do tipo FULL\_STAT (fSt) e os semáforos que irão ser utilizados para evitar situações de deadlock, que são:

* mutex (valor inicial um) – este semáforo é usado para identificar a entrada na região crítica e a saída da mesma. Uma região crítica é uma zona de código que manipula dados partilhados e que não pode ser executada concorrentemente por mais do que um processo;
* friendsArrived (valor inicial zero) – identifica o semáforo usado pelos clientes para esperarem que os amigos cheguem;
* requestReceived (valor inicial zero) – este semáforo é usado pelos clientes para esperar pelo empregado depois de este ter feito um pedido;
* foodArrived (valor inicial zero) – identifica o semáforo usado pelos clientes para esperarem que a comida chegue;
* allFinished (valor inicial zero) – este semáforo é usado pelos clientes para esperarem que cada um termine a sua refeição;
* waiterRequest (valor inicial zero) – identifica o semáforo usado pelo empregado para esperar por um pedido vindo ou de um cliente ou do chefe;
* waitOrder (valor inicial zero) – este semáforo é usado pelo chefe para esperar por um pedido.

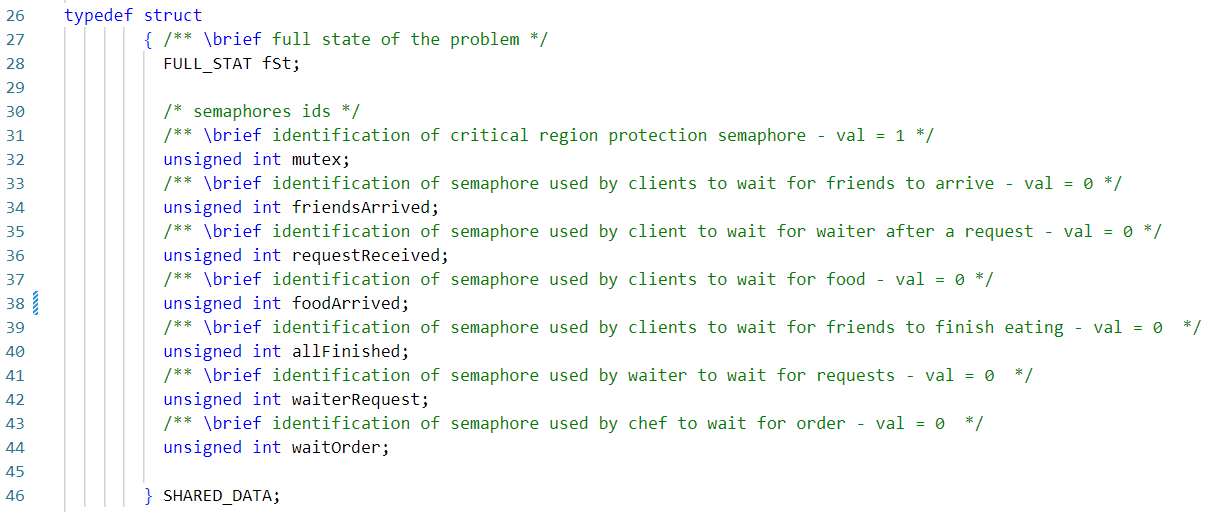


Fig.3 –Estrutura definida no ficheiro sharedDataSync.h.