Sistemas Operativos

2022/2023

2ºProjeto

Jantar de Amigos

Hugo Correia 108215 P6 50%

Sara Almeida 108796 P6 50%

Professor:

Nuno Lau (nunolau@ua.pt)

Índice

Introdução 3

Introdução ao Problema 4

Scripts Disponibilizados 4

O Problema 5

Implementação 7

Waiter 7

waiterStat 7

waitForClientOrChef() 7

informChef() 9

takeFoodToTable() 9

receivePayment() 10

Chef 11

chefStat 11

waitForOrder() 11

processOrder() 12

Client 13

clientStat[id] 13

waitFriends() 13

orderFood() 15

waitFood() 16

waitAndPay() 17

Diagrama Resumo 19

Resultados Obtidos 20

Conclusão 23

Bibliografia 23

1 – Introdução

No contexto da disciplina de Sistemas Operativos, foi-nos proposta a realização deste projeto que tem como objetivo a compreensão dos mecanismos associados à execução e sincronização de processos e *threads*.

Para a realização deste trabalho foi utilizado o **VSCODE**, um editor conhecido por ambos os elementos deste grupo e com o qual estamos habituados a trabalhar no seguimento de outras disciplinas. A execução de todo o trabalho foi sendo suportada por um repositório de GitHub, para facilitar a sua manipulação por ambos os elementos do grupo e para serem salvas cópias de segurança caso haja algum erro ao longo do processo do trabalho.

O problema é baseado num jantar de amigos (*clients*) organizado num restaurante com um empregado de mesa (*waiter*) e um chefe de cozinha (*chef*). São impostas algumas condições no enunciado que influenciam a forma de resolução do problema, entre elas, que o primeiro amigo a chegar deve fazer todo o pedido, no fim de todos os restantes amigos chegarem, e o último a chegar fica com a responsabilidade de pagar a conta, depois de todos terem terminado a refeição. O principal desafio é coordenar todos estes processos independentes utilizando semáforos.

Este projeto foi imensamente vantajoso na compreensão de processos e *threads*, bem como da sua manipulação e, principalmente, sincronização. Deste modo, a realização deste trabalho permitiu o alargamento dos nossos conhecimentos e clarificou substancialmente a nossa perceção sobre a matéria de semáforos e sua implementação.

2 – Introdução ao Problema

2.1 – Scripts Disponibilizados

Para a realização deste projeto, foram-nos inicialmente disponibilizadas 3 pastas com vários ficheiros, de entre os quais modificámos apenas 4.

Na pasta **doc**, encontra-se um ficheiro "*Doxyfile*" com todas as configurações usadas pelo sistema doxygen para o projeto.

Na pasta **run** encontram-se os ficheiros com os quais o utilizador irá entrar em contacto sempre que queira interagir com o projeto através do prompt. Estão presentes ficheiros em bash-shell como *run.sh*, onde o programa irá correr o programa probSemSharedMemRestaurant que contém todos os restantes ficheiros e funções alteradas por nós para que o programa corra corretamente. Caso esse script seja corrido sem argumentos, serão gerados 1000 restaurantes sendo que cada um terá as caraterísticas já acima definidas. Caso

contrário serão gerados o número de restaurantes que o utilizador definir.

Em particular, o ficheiro *clean.sh* foi modificado, mais especificamente as respetivas chaves de semáforo e shared memory para os valores das mesmas, mas em cada um dos nossos dispositivos. Como resultado da execução deste script são limpos os erros deixados ao correr o programa e é limpa a memória deixada em cache pelo mesmo. Por fim, foram dados pelo professor os programas de cada uma das entidades já compilados para que seja mais fácil a compreensão dos resultados que se pretendem obter.

Finalmente, na pasta **src** é onde está toda a informação relativa ao projeto, de entre a qual, se destacam alguns ficheiros. No ficheiro *probConst.h* é onde são definidas algumas constantes como a TABLESIZE, MAXEAT e MAXCOOK e os estados de cada um dos participantes. No ficheiro *probDataStruct.h* estão definidos os estados dos intervenientes dentro de uma data do tipo *struct*, seguidos de algumas *flags* e variáveis importantes relacionadas com os mesmo. No ficheiro *probSemSharedMemRestaurant.c* é onde se encontra a main, onde todos os semáforos e memórias são sincronizados e onde é iniciada a simulação gerando, então, as entidades *chef*, *client* e *waiter*. Também importante é o ficheiro *sharedDataSync.h* que contém, em particular, as identificações dos semáforos.

Nos ficheiros *sharedMemory.h* e *sharedMemory.c* é tratado o domínio da memória partilhada e nos ficheiros *semaphore.c* e *semaphore.h* são manipulados os semáforos.

Por fim, ainda dentro da pasta **src**, são definidos os três programas nos quais foram feitas as mudanças pela nossa parte, colocando em prática e funcionamento o projeto, sendo estes os ficheiros *semSharedMemChef.c*, *semSharedMemClient.c* e *semSharedMemWaiter.c.* Cada um destes ficheiros tem já criada uma função *main* e as chamadas às funções completadas por nós na ordem devida.

2.2 – O problema

Neste projeto todas as entidades, ou seja, o *chef*, o *waiter* e os *clients* são processos independentes. No entanto, este conjunto de processos, para simular o jantar de amigos e toda a sua dinâmica, terão de correr ao mesmo tempo e, por sua vez, estes partilham uma ou mais variáveis manipulando as mesmas de forma independente entre si. Assim, é necessário haver algum mecanismo de sincronização, para que os resultados obtidos sejam os esperados e para que a dinâmica do problema faça sentido.

Como solução, usaremos então **semáforos**, imensamente úteis na sincronização de processos, auxiliando a comunicação entre os mesmos.

Para conseguirmos obter uma sicronização correta temos de ter em conta as exigências ou regras dadas pelo enunciado: o primeiro amigo a chegar faz o pedido, depois de todos chegarem; o *waiter* leva o pedido ao *chef* e recolhe-o quando estiver pronto para levar à mesa; o último amigo a ter chegado paga a conta quando todos tiverem terminado a sua refeição e o tamanho da mesa é ajustado ao número de amigos.

Para facilitar o processo de implementação e para melhor o explicarmos, criámos uma tabela, que pode ser vista abaixo, com todos os semáforos, incluindo a zona crítica, utilizados.

Semáforo	Down			Up		
	Entidade	Função	Nº Downs	Entidade	Função	Nº Ups
Mutex	Client*1	WaitFriends	1	Client*1	WaitFriends	1
		waitFood	2		waitFood	2
	Client (exceto lastClient)	waitAndPay	2	Client (exceto lastClient)	waitAndPay	2
	firstClient	orderFood	1	firstClient	orderFood	1
	lastClient	waitAndPay	3	lastClient	waitAndPay	3
	Chef	waitForOrder	1	Chef	waitForOrder	1
		processOrder	1		processOrder	1
	Waiter	waitForClientOrChef	6	Waiter	waitForClientOrChef	6
		informChef	1		informChef	1
		takeFoodToTable	1		takeFoodToTable	1
		receivePayment	1		receivePayment	1
friendsArrived	Client	waitFriends	1	lastClient	waitFriends	1
	firstClient		1			
requestReceived	firstClient	orderFood	1	Waiter	informChef	1
	lastClient	waitAndPay	1		receivePayment	1
foodArrived	Client	waitFood	1	Waiter	takeFoodToTable	TABLESIZE
waiterRequest	Waiter	waitForClientOrChef	3	firstClient	orderFood	1
				lastClient	waitAndPay	1
				Chef	processOrder	1
waitOrder	Chef	waitForOrder	1	Waiter	informChef	1
allFinished	Client	waitAndPay	1	Último Client (a acabar de comer)	WaitAndPay	TABLESIZE

*1 – Todas as funções pertencentes ao Cliente também incluem o lastClient e o firstClient, exceto aquando os mesmo são referidos como Entidades no mesmo semáforo.

3 – Implementação

3.1. – Waiter

3.1.1 – waiterStat

waiterStat	
WAIT_FOR_REQUEST 0	O *waiter* espera por pedido de comida
INFORM_CHEF 1	O *waiter* leva o pedido de comida ao *chef*
TAKE_TO_TABLE 2	O *waiter* leva a comida à mesa
RECEIVE_PAYMENT 3	O *waiter* recebe pagamento

3.1.2 – Função waitForClientOrChef()

```
static int waitForClientOrChef()
{
    int ret=0;
    if (semDown (semgid, sh->mutex) == -1)  {
        perror ("error on the up operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }

    /* insert your code here */
    sh->fSt.st.waiterStat = WAIT_FOR_REQUEST;
    saveState(nFic, &(sh->fSt));


    if (semUp (semgid, sh->mutex) == -1)      {
        perror ("error on the down operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
```

Figura 1 - Função waitForClientOrChef()

Nesta função, o *waiter* espera pelo próximo pedido, seja ele do cliente ou do chef, por isso atualizámos o seu estado para WAIT_FOR_REQUEST, dentro da região crítica do *mutex*, guardando o mesmo (Fig.1).

```
/* insert your code here */
if (semDown (semgid, sh->waiterRequest) == -1)  {
    perror ("error on the up operation for semaphore access (WT)");
    exit (EXIT_FAILURE);
}

if (semDown (semgid, sh->mutex) == -1)  {
    perror ("error on the up operation for semaphore access (WT)");
    exit (EXIT_FAILURE);
}

/* insert your code here */
if (sh->fSt.foodRequest == 1) {
    ret = FOODREQ;
    sh->fSt.foodRequest = 0;
} else if (sh->fSt.foodReady == 1) {
    sh->fSt.foodReady = 0;
    ret = FOODREADY;
} else if (sh->fSt.paymentRequest == 1) {
    sh->fSt.paymentRequest = 0;
    ret = BILL;
}

if (ret == '\0'){
    perror ("ret!=0 semSharedMemWaiter.c waitForClientOrChef");
    exit (EXIT_FAILURE);
}

if (semUp (semgid, sh->mutex) == -1) {
 perror ("error on the down operation for semaphore access (WT)");
    exit (EXIT_FAILURE);
}

return ret;
```

Figura 2 - Função waitForClientOrChef() (continuação)

No entanto, como podemos observar acima na Fig.2, em particular, esta função terá de retornar uma valor *ret*, inicializado como 0, que irá determinar a próxima função do script a ser executada, dependendo do tipo de pedido e de quem fez o mesmo. Assim, por exemplo, caso a *flag* foodRequest esteja ativada significa que o *client* fez o pedido de refeição, logo, **ret** = FOODREQ. Consequentemente, o ciclo de vida do *waiter* define que, quando ret assume esse valor, a seguinte função a ser executada é a **informChef().** E assim igualmente para os restantes valores possíveis de **ret** (FOODREADY e BILL).

Este conjunto de condições *if* que define o valor de **ret** é executado dentro da região crítica do *mutex*, depois de se fazer *semDown* ao semáforo *waiterRequest.*

3.1.3 – Função informChef()

```
static void informChef ()
{
    if (semDown (semgid, sh->mutex) == -1)  {
        perror ("error on the up operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }

    /* insert your code here */
    sh->fSt.foodOrder = 1;
    sh->fSt.st.waiterStat = INFORM_CHEF;
    saveState(nFic, &(sh->fSt));

    if (semUp (semgid, sh->mutex) == -1)
    { perror ("error on the down operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }

    /* insert your code here */
    if (semUp (semgid, sh->requestReceived) == -1) {
        perror ("error on the up operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }
    if (semUp (semgid, sh->waitOrder) == -1) {
        perror ("error on the up operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }
}
```

Figura 3 - Função informChef()

Nesta função (Fig.3), o *waiter* leva o pedido de refeição ao *chef,* por isso, dentro da região crítica do *mutex*, ativamos a *flag* foodOrder e atualizamos o seu estado para INFORM_CHEF e guardamo-lo. Para além disto, fazemos *semUp* dos semáforos *requestReceived* e *waitOrder,* semáforos estes usados, respetivamente, pelo *client* para esperar pelo *waiter* depois de um pedido e pelo *chef* para esperar pelo pedido de refeição trazido pelo *waiter,* pois estes estados já terminaram.

3.1.4 – Função takeFoodToTable()

```
static void takeFoodToTable ()
{
    if (semDown (semgid, sh->mutex) == -1)  {
        perror ("error on the up operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }

    /* insert your code here */
    sh->fSt.st.waiterStat = TAKE_TO_TABLE;
    saveState(nFic, &(sh->fSt));
    for (int i = 0; i < sh->fSt.tableClients; i++) {
        if (semUp (semgid, sh->foodArrived) == -1) {
            perror ("error on the up operation for semaphore access (WT)");
            exit (EXIT_FAILURE);
        }
    }
    if (semUp (semgid, sh->mutex) == -1)  {
     perror ("error on the down operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }
}
```

Figura 4 - Função takeFoodToTable()

Na função representada (Fig.4), o *waiter* leva a comida para a mesa, no entanto, é importante verificar que todos os amigos da mesa recebem comida. Assim, dentro da região crítica do *mutex*, o *waiter* vai ter o seu estado atualizado para TAKE_TO_TABLE e guardado. De seguida, utilizámos um pequeno ciclo *for* que percorre o número de clientes da mesa, utilizando a variável *tableClients* e , para cada um, dá *semUp* do semáforo *foodArrived*, para conseguirmos certificar-nos que este processo foi realizado no mesmo número de clientes da mesa, indicando que todos receberam comida e podem, de seguida, começar a sua refeição.

3.1.5 – Função receivePayment()

```
static void receivePayment ()
{
    if (semDown (semgid, sh->mutex) == -1)  {
        perror ("error on the up operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }

    /* insert your code here */
    sh->fSt.st.waiterStat=RECEIVE_PAYMENT;
    saveState (nFic, &sh->fSt);

    if (semUp (semgid, sh->requestReceived) == -1) {
        perror ("error on the up operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }

    if (semUp (semgid, sh->mutex) == -1)  {
     perror ("error on the down operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }
}
```

Figura 5 - Função receivePayment()

Nesta função (Fig.5), após todos os clientes acabarem a sua refeição procedem com o pagamento junto do *waiter*. Este recebe o pagamento do último *client* que chegou (definido nas funções Client) atualizando e guardando o seu estado para RECEIVE_PAYMENT, dentro da região critica do *mutex*. Após a efetuação do pagamento, fazemos semUp do semáforo *requestReceived* (já explicado acima). Sendo que agora o *waiter* deixou de estar a espera por pedidos, liberta o espaço na memória ocupada pela mesmo e para de correr.

3.2. – Chef

3.2.1 – chefStat

chefStat	
WAIT_FOR_ORDER 0	O *chef* espera por pedido de comida
COOK 1	O *chef* está a cozinhar
REST 2	O *chef* está a descansar

3.2.2 – Função waitForOrder()

```
static void waitForOrder ()
{
    /* insert your code here */
    // sh->fSt.st.chefStat = WAIT_FOR_ORDER;
    // saveState(nFic, &(sh->fSt));
    if (semDown(semgid, sh->waitOrder) == -1) {
        perror("error on the down operation for semaphore access (PT)");
        exit(EXIT_FAILURE);
    }

    if (semDown (semgid, sh->mutex) == -1) {
        perror ("error on the up operation for semaphore access (PT)");
        exit (EXIT_FAILURE);
    }

    /* insert your code here */
    sh->fSt.foodOrder = 0;
    sh->fSt.st.chefStat = COOK;
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the up operation for semaphore access (PT)");
        exit (EXIT_FAILURE);
    }
}
```

Figura 6 - Função waitForOrder()

Inicialmente, nesta função (Fig.6) utilizamos o semáforo *waitOrder* (semáforo usado pelo *chef* quando o mesmo está à espera de algum pedido) e entramos na região crítica do *mutex*. Dentro desta última, é atualizado o valor da *flag* foodOrder para 0, ou seja, após a conclusão de um pedido, o *chef* ficará de novo à espera do próximo e daí em diante. Mal receba esse novo pedido atualiza o seu estado para COOK, começando, portanto, a preparar o respetivo pedido, guarda o seu novo estado e sai da zona crítica.

3.2.3 – Função processOrder()

```
static void processOrder ()
{
    usleep((unsigned int) floor ((MAXCOOK * random ()) / RAND_MAX + 100.0));

    if (semDown (semgid, sh->mutex) == -1) {
        perror ("error on the up operation for semaphore access (PT)");
        exit (EXIT_FAILURE);
    }

    /* insert your code here */
    sh->fSt.foodReady = 1;
    sh->fSt.st.chefStat = REST;
    saveState(nFic, &(sh->fSt));


    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the up operation for semaphore access (PT)");
        exit (EXIT_FAILURE);
    }

    /* insert your code here */
    if (semUp(semgid, sh->waiterRequest) == -1) {
        perror("error on the up operation for semaphore access (PT)");
        exit(EXIT_FAILURE);
    }

}
```

Figura 7 - Função processOrder()

Nesta função (Fig.7), o *chef* continua a cozinhar o pedido que começou na função acima descrita sendo que o tempo que demorará a estar pronto será definido pelo *usleep*, logo no início da função. Este terá um valor aleatório respeitando uma *distribuição normal.*

De seguida, entra na região crítica do *mutex* e, após terminar a confeção da refeição aciona a *flag* foodReady, sendo a mesma resgatada mais tarde pelo *waiter* na função ***waitForClientOrChef()*** onde o mesmo, sabendo que a comida está pronto procede, então, com a recolha da mesma. Ainda antes da recolha da comida por parte do *waiter* já o *chef* atualiza o seu estado para REST, guarda-o, sai da zona crítica do *mutex* e fazemos *semUp* no semáforo *waiterRequest* (o waiter deixa de estar à espera do cozinheiro).

3.3. – Client

3.3.1 – clientStat[id]

clientStat[id]	
INIT 1	Estado inicial do *client*
WAIT_FOR_FRIENDS 2	O *client* espera que os restantes amigos cheguem
FOOD_REQUEST 3	O *client* faz o pedido ao *waiter*
WAIT_FOR_FOOD 4	O *client* espera pela comida
EAT 5	O *client* está a comer
WAIT_FOR_OTHERS 6	O *client* está à espera que os amigos acabem a refeição
WAIT_FOR_BILL 7	O *client* está à espera para completar o pagamento
FINISHED 8	O *client* termina a sua refeição

3.3.2 – Função waitFriends(*int id*)

```
static bool waitFriends(int id)
{
    bool first = false;

    if (semDown (semgid, sh->mutex) == -1) {
        perror ("error on the down operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
    }

    /* insert your code here */
    sh->fSt.tableClients++;

    if(sh->fSt.tableClients == 1){
        first = true;
        sh->fSt.tableFirst = id;
    }

    if(sh->fSt.tableClients == TABLESIZE){
        sh->fSt.tableLast = id;
        sh->fSt.st.clientStat[id] = WAIT_FOR_FOOD;
        saveState(nFic, &(sh->fSt));
        for(int i = 0; i <= TABLESIZE - 1; i++){
            if (semUp (semgid, sh->friendsArrived) == -1) {
                perror ("error on the down operation for semaphore access (CT)");
                exit (EXIT_FAILURE);
            }
        }

    }
    else{
        sh->fSt.st.clientStat[id] = WAIT_FOR_FRIENDS;
        saveState(nFic, &(sh->fSt));
    }


    if (semUp (semgid, sh->mutex) == -1)
    { perror ("error on the up operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
    }
```

Figura 8 - Função waitFriends()

A função observada acima (Fig.8) simula a espera do *client* até a mesa estar completa mas, para além disso, tem o objetivo de registar os *id's* dos primeiro e último *client,* dado que estes têm responsabilidades diferentes dos demais, e de garantir que os amigos esperam todos um pelos outros, completando a mesa. Para auxiliar no cumprimento destas funcionalidades iniciámos uma variável **bool** *first* com o valor false, que será retornada no final da função.

Assim, dentro da região crítica do *mutex,* começamos por incrementar o número de amigos representado pela variável *tableClients*, pois cada vez que a função é executada significa que um novo amigo chegou. De seguida, através de uma condição *if* que verifica se *tableClients* é igual a 1, indicando que se trata do primeiro *client,* alteramos o valor da variável *first* para true e guardamos o *id* do mesmo na variável *tableFirst.* Seguindo a mesma lógica, quando *tableClients* tiver valor igual a TABLESIZE (tamanho da mesa), significa que este será o último cliente e, por isso, guardamos o seu *id* na variável *tableLast* . Dentro ainda desta condição *if,* usámos um ciclo *for* que percorre o número de amigos, ou seja, o TABLESIZE, para todos os amigos fazerem *semUp* do semáforo *friendsArrived,* indicando que a espera de todos os amigos pelos demais já terminou. Caso contrário, ou seja, enquanto todos os amigos não chegarem, alteramos o estado de cada *client*, exceto do último portanto, para WAIT_FOR_FRIENDS e guardamo-lo.

```
    /* insert your code here */
    if (sh->fSt.tableClients != TABLESIZE){
        if (semDown (semgid, sh->friendsArrived) == -1){
            perror ("error on the up operation for semaphore access (CT)");
            exit (EXIT_FAILURE);
            }
        }

    return first;
}
```

Figura 9 - Função waitForFriends() (continuação)

Por fim, inserimos uma condição *if* (Fig.9) que será executada enquanto *tableClients* != TABLESIZE, ou seja, enquanto não chegarem todos os amigos, que faz *semDown* do semáforo *friendsArrived,* indicando que a espera de uns amigos pelos outros permanece, o que só irá mudar nas circunstâncias já explicadas acima.

3.3.3 – Função orderFood(*int id)*

```
static void orderFood (int id)
{
    if (semDown (semgid, sh->mutex) == -1) {
        perror ("error on the down operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
    }

    /* insert your code here */
    sh->fSt.foodRequest = 1;
    if (semUp (semgid, sh->waiterRequest) == -1)
    { perror ("error on the up operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
    }
    sh->fSt.st.clientStat[sh->fSt.tableFirst] = FOOD_REQUEST;
    saveState(nFic, &(sh->fSt));

    if (semUp (semgid, sh->mutex) == -1)
    { perror ("error on the up operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
    }

    /* insert your code here */
    if (semDown(semgid, sh->requestReceived) == -1) {
    perror("error on the up operation for semaphore access(GL)");
    exit(EXIT_FAILURE);
    }
}
```

Figura 10 - Função orderFood()

Esta função é apenas inicializada com o *id* do primeiro *client* a chegar ao restaurante (Fig.10), pois só este pode fazer o pedido de acordo com o enunciado. Inicialmente, entramos na região crítica do *mutex* onde a *flag* foodRequest será acionada para, novamente, ser mais tarde chamada pela função **waitForClientOrChef()** onde, eventualmente, o waiter procederá à recolha desse mesmo pedido. Logo de seguida e seguindo a lógica da função anterior referida, o *waiter* deixará de estar à espera que o *client* faça o seu pedido, logo fazemos *semUp* do semáforo do *waiterRequest*.

O estado do primeiro *client* é então atualizado e guardado como FOOD_REQUEST, onde o mesmo começa, obedecendo ao cenário, a fazer o pedido ao *waiter*. Por fim, saímos da zona crítica do *mutex* e os clientes começam a espera pela sua comida, ou seja, ficam novamente à espera do *waiter*, o que é simulado fazendo *semDown* do semáforo *requestReceived*.

3.3.4 – Função waitFood(*int id)*

```
static void waitFood (int id)
{

    if (semDown (semgid, sh->mutex) == -1) {
        perror ("error on the down operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
    }

    /* insert your code here */
    sh->fSt.st.clientStat[id] = WAIT_FOR_FOOD;
    saveState(nFic, &(sh->fSt));

    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the down operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
    }

    /* insert your code here */
    if (semDown (semgid, sh->foodArrived) == -1) {
        perror ("error on the down operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
    }

    if (semDown (semgid, sh->mutex) == -1) {
        perror ("error on the down operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
    }

    /* insert your code here */
    sh->fSt.st.clientStat[id] = EAT;
    saveState(nFic, &(sh->fSt));

    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the down operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
    }
}
```

Figura 11 - Função waitFood()

Esta função (Fig.11) é inicializada com o id de cada um dos clientes, sendo que começa por darmos entrada na região crítica do *mutex*. De seguida, como cada cliente ainda se encontra a espera da comida, após o pedido ter sido efetuado na função anterior, o seu estado será alterado para WAIT_FOR_FOOD. Portanto, saímos da zona crítica do *mutex* e fazemos *semDown* no semáforo *foodArrived* (semáforo indicador da espera pela comida pelo *client*).

Após o *waiter* entregar a comida a cada um dos clientes na função **takeFoodToTable()**, voltamos a entrar na região critica e cada um dos clientes, já com o prato na mesa, muda o seu estado para EAT e guarda-o, simulando o início da sua refeição.

3.3.5 – Função waitAndPay(*int id)*

```
static void waitAndPay (int id)
{
    bool last=false;

    if (semDown (semgid, sh->mutex) == -1) {
        perror ("error on the down operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
    }


    /* insert your code here */
    last = (sh->fSt.tableLast == id);
    sh->fSt.st.clientStat[id] = WAIT_FOR_OTHERS;
    saveState(nFic, &(sh->fSt));

    sh->fSt.tableFinishEat++;
    if (sh->fSt.tableFinishEat == TABLESIZE){
        for (int i = 0; i < TABLESIZE; i++){
            if (semUp (semgid, sh->allFinished) == -1) {
                perror ("error on the down operation for semaphore access (CT)");
                exit (EXIT_FAILURE);
            }
        }
    }


    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the down operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
    }

    /* insert your code here */
    if (semDown (semgid, sh->allFinished) == -1) {
        perror ("error on the down operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
    }
```

Figura 12 - Função waitAndPay()

Esta função (Fig.12), simula a espera dos amigos para que todos acabem a sua refeição e o pagamento que apenas pode ser efetuado pelo último amigo a ter chegado ao jantar. Para auxiliar ao cumprimento desta última exigência, foi iniciada uma variável **bool** *last* com o valor false.

Começamos por dar entrada na região crítica do *mutex*. Aqui, é inicialmente atualizado o valor da variável *last* que apenas tomará o valor true quando o *id* do *client* que está a ativar a execução da função for igual ao guardado na *tableLast,* ou seja, quando este for o último amigo a ter chegado.

De seguida, o estado cada *client* é atualizado para WAIT_FOR_OTHERS e guardado, simulando que espera que os restantes amigos terminem a sua refeição.

Prosseguindo, sempre que esta função for chamada, a variável *tableFinishEat* é incrementada, pois é neste momento da simulação que o *client* vai terminar a sua refeição. Logo, em semelhança ao que acontece na função ***waitFriends()***, com uma condição *if* e um ciclo *for,* quando o valor de *tableFinishEat* for igual ao TABLESIZE, fazemos *semUp* do semáforo *allFinished* em número igual ao número de amigos, simulando que todos terminaram a sua refeição e esperam que os restantes amigos o façam.

Assim, podemos então sair da zona crítica do *mutex* e fazer *semDown* do semáforo *allFinished.*

```
if(last) {
    if (semDown (semgid, sh->mutex) == -1) {
       perror ("error on the down operation for semaphore access (CT)");
       exit (EXIT_FAILURE);
    }

    /* insert your code here */
    sh->fSt.st.clientStat[id] = WAIT_FOR_BILL;
    saveState(nFic, &sh->fSt);
    sh->fSt.paymentRequest = 1;

    if (semUp (semgid, sh->waiterRequest) == -1) {
        perror ("error on the down operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
    }

    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the down operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
    }

    /* insert your code here */
    if (semDown (semgid, sh->requestReceived) == -1) {
        perror ("error on the down operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
    }
    //semDown(semgid, sh->requestReceived);
}
```

Figura 13 - Função waitAndPay() (continuação)

Como o último amigo a chegar tem a responsabilidade de pagar, temos de garantir essa funcionalidade para ele apenas, tendo tal sido conseguido pela condição *if* (Fig.13) que apenas será executada quando, como explicado acima, a variável **bool** *last* tiver o valor true.

Dentro dessa condição, entramos novamente na região crítica do *mutex,* onde o estado deste *client* em particular vai ser atualizado para WAIT_FOR_BILL e guardado como tal. Aqui, ativamos também a *flag* paymentRequest, usada pelo último para fazer o pedido de pagamento ao *waiter.* Finalmente, podemos fazer *semUp* do semáforo waiterRequest, pois o *waiter* já terá sido chamado para a efetuação do pagamento, e sair da região crítica , bem como fazer *semDown* do semáforo *requestReceived,* simulando a espera do *client* pelo *waiter* depois de lhe ter feito o pedido.

```
if (semDown (semgid, sh->mutex) == -1) {
    perror ("error on the down operation for semaphore access (CT)");
    exit (EXIT_FAILURE);
}

/* insert your code here */
sh->fSt.st.clientStat[id] = FINISHED;
saveState(nFic, &(sh->fSt));

if (semUp (semgid, sh->mutex) == -1) {
    perror ("error on the down operation for semaphore access (CT)");
    exit (EXIT_FAILURE);
}
```

Figura 14 - Função waitAndPay() (continuação)

Para finalizar esta função (Fig.14), novamente dentro da região crítica do *mutex,* é atualizado o estado de cada client para FINISHED e guardado como tal, terminando assim o jantar de amigos.

3.4. – Diagrama Resumo

Através do diagrama abaixo apresentado, podemos observar a dinâmica deste jantar, tendo em conta os estados de cada interveniente e a sua ordem de funcionamento, como forma de complemento e sumarização de tudo acima explicado.

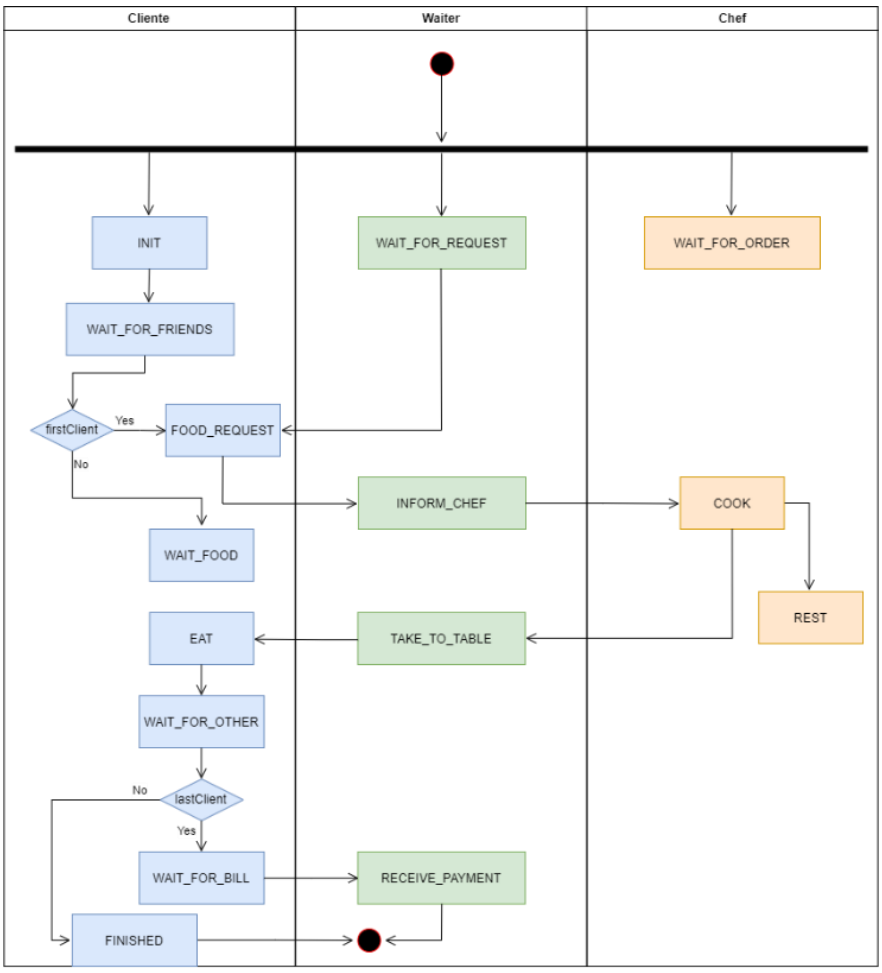


Figura 15 - Diagrama de estados

4 – Resultados Obtidos

4.1. – Resultados Professor

Para este projeto, na pasta run, foram-nos disponibilizados ficheiros compilados que quando corridos imprimem no terminal a solução do problema.

Para isso, no diretório semaphore_restaurant/src corremos o comando *make all_bin* e, de seguida, no diretório semaphore_restaurant/run corremos o programa fazendo *./run 1,* o que, como visto na introdução é equivalente a fazer *./probSemSharedMemRestaurant.*

4.2. – Resultados Alunos

Para correr a nossa implementação e resolução do problema, no diretório semaphore_restaurant/src corremos o comando *make* e, de seguida, no diretório semaphore_restaurant/run corremos o programa fazendo *./run 1.*

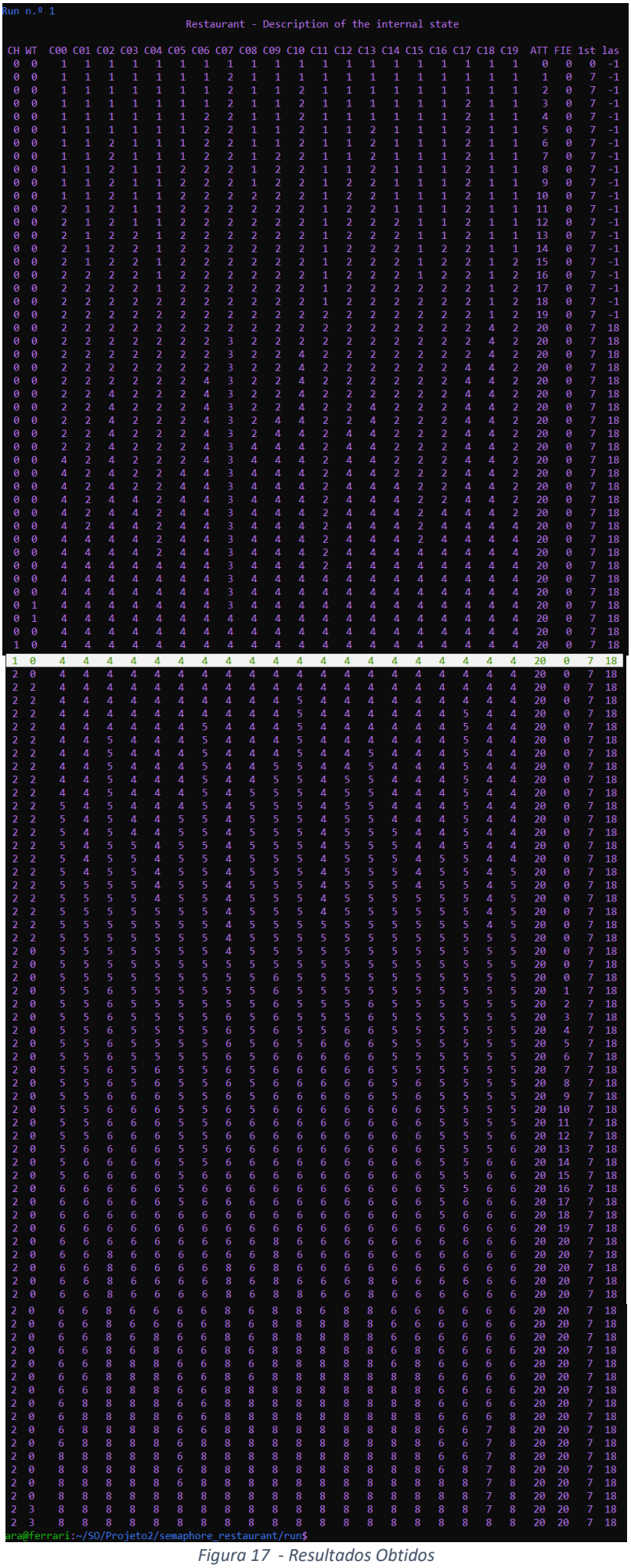
Nas duas páginas seguintes, podemos observar os resultados obtidos sendo que na primeira página está o resultado da solução disponibilizada e, na página seguinte, os resultados obtidos por nós.

Analisando e comparando ambos, achamos que realizámos um bom trabalho e sucedemos na implementação dos semáforos e respeito das regras dadas pelo enunciado.

```
un n.º 1
                                Restaurant - Description of the internal state

CH WT  C00 C01 C02 C03 C04 C05 C06 C07 C08 C09 C10 C11 C12 C13 C14 C15 C16 C17 C18 C19  ATT FIE 1st las
 0  0    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1    0   0   0  -1
 0  0    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   2   1   1   1    1   0  16  -1
 0  0    1   1   1   1   1   2   1   1   1   1   1   1   1   1   1   1   2   1   1   1    2   0  16  -1
 0  0    1   1   1   1   1   2   1   1   1   1   1   1   1   1   1   1   2   1   1   2    3   0  16  -1
 0  0    1   1   1   1   1   2   1   1   2   1   1   1   1   1   1   1   2   1   1   2    4   0  16  -1
 0  0    1   1   1   1   1   2   1   1   2   1   1   1   1   1   1   2   2   1   1   2    5   0  16  -1
 0  0    1   1   1   1   2   2   1   1   2   1   1   1   1   1   1   2   2   1   1   2    6   0  16  -1
 0  0    1   1   1   1   2   2   1   1   2   1   1   2   1   1   1   2   2   1   1   2    7   0  16  -1
 0  0    1   1   1   1   2   2   1   1   2   1   1   2   1   1   1   2   2   1   2   2    8   0  16  -1
 0  0    2   1   1   1   2   2   1   1   2   1   1   2   1   1   1   2   2   1   2   2    9   0  16  -1
 0  0    2   1   1   1   2   2   1   1   2   1   2   2   1   1   1   2   2   1   2   2   10   0  16  -1
 0  0    2   1   1   1   2   2   2   1   2   1   2   2   1   1   1   2   2   1   2   2   11   0  16  -1
 0  0    2   1   1   1   2   2   2   1   2   1   2   2   1   2   1   2   2   1   2   2   12   0  16  -1
 0  0    2   1   2   1   2   2   2   1   2   1   2   2   1   2   1   2   2   1   2   2   13   0  16  -1
 0  0    2   1   2   1   2   2   2   1   2   2   2   2   1   2   1   2   2   1   2   2   14   0  16  -1
 0  0    2   1   2   1   2   2   2   1   2   2   2   2   2   2   1   2   2   1   2   2   15   0  16  -1
 0  0    2   2   2   1   2   2   2   1   2   2   2   2   2   2   1   2   2   1   2   2   16   0  16  -1
 0  0    2   2   2   1   2   2   2   2   2   2   2   2   2   2   1   2   2   1   2   2   17   0  16  -1
 0  0    2   2   2   1   2   2   2   2   2   2   2   2   2   2   2   2   2   1   2   2   18   0  16  -1
 0  0    2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   1   2   2   19   0  16  -1
 0  0    2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   4   2   2   20   0  16  17
 0  0    2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   3   4   2   2   20   0  16  17
 0  0    2   2   2   2   2   4   2   2   2   2   2   2   2   2   2   2   3   4   2   2   20   0  16  17
 0  0    2   2   2   2   2   4   2   2   4   2   2   2   2   2   2   2   3   4   2   2   20   0  16  17
 0  0    2   2   2   2   2   4   2   2   4   2   2   2   2   2   2   2   3   4   2   2   20   0  16  17
 0  0    2   2   2   2   2   4   2   2   4   2   2   2   2   2   2   4   3   4   2   4   20   0  16  17
 0  0    2   2   2   2   4   4   2   2   4   2   2   2   2   2   2   4   3   4   2   4   20   0  16  17
 0  0    2   2   2   2   4   4   2   2   4   2   2   4   2   2   2   4   3   4   2   4   20   0  16  17
 0  0    2   2   2   2   4   4   2   2   4   2   2   4   2   2   2   4   3   4   4   4   20   0  16  17
 0  0    4   2   2   2   4   4   2   2   4   2   2   4   2   2   2   4   3   4   4   4   20   0  16  17
 0  0    4   2   2   2   4   4   2   2   4   2   4   4   2   2   2   4   3   4   4   4   20   0  16  17
 0  0    4   2   2   2   4   4   4   2   4   2   4   4   2   2   2   4   3   4   4   4   20   0  16  17
 0  0    4   2   4   2   4   4   4   2   4   2   4   4   2   2   2   4   3   4   4   4   20   0  16  17
 0  0    4   2   4   2   4   4   4   2   4   2   4   4   2   4   2   4   3   4   4   4   20   0  16  17
 0  0    4   2   4   2   4   4   4   2   4   2   4   4   2   4   2   4   3   4   4   4   20   0  16  17
 0  0    4   2   4   2   4   4   4   2   4   4   4   4   2   4   2   4   3   4   4   4   20   0  16  17
 0  0    4   4   4   2   4   4   4   2   4   4   4   4   4   4   2   4   3   4   4   4   20   0  16  17
 0  0    4   4   4   2   4   4   4   2   4   4   4   4   4   4   2   4   3   4   4   4   20   0  16  17
 0  0    4   4   4   2   4   4   4   4   4   4   4   4   4   4   4   4   3   4   4   4   20   0  16  17
 0  0    4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   3   4   4   4   20   0  16  17
 0  1    4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   3   4   4   4   20   0  16  17
 0  1    4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0  16  17
 0  0    4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0  16  17
 1  0    4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0  16  17
 1  0    4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0  16  17
 2  0    4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0  16  17
 2  2    4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0  16  17
 2  2    4   4   4   4   4   4   5   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0  16  17
 2  2    4   4   4   4   4   4   5   4   4   5   4   4   4   4   4   4   4   4   4   4   20   0  16  17
 2  2    4   4   4   4   4   4   5   4   4   5   4   4   4   4   4   4   5   4   4   4   20   0  16  17
 2  2    4   4   4   4   4   5   5   4   4   5   4   4   4   4   4   4   5   4   4   4   20   0  16  17
 2  2    4   4   4   4   5   5   4   4   5   4   4   5   4   4   4   5   4   4   4   5   20   0  16  17
 2  2    5   4   4   4   5   5   4   4   5   4   4   5   4   4   4   5   4   4   5   5   20   0  16  17
 2  2    5   4   4   4   5   5   4   4   5   4   5   5   4   4   4   5   4   4   5   5   20   0  16  17
 2  2    5   4   4   4   5   5   5   4   5   4   5   5   4   4   4   5   4   4   5   5   20   0  16  17
 2  2    5   4   5   4   5   5   5   4   5   4   5   5   4   4   4   5   4   4   5   5   20   0  16  17
 2  2    5   4   5   4   5   5   5   4   5   4   5   5   4   5   4   5   4   4   5   5   20   0  16  17
 2  2    5   4   5   4   5   5   5   5   5   4   5   5   5   5   4   5   4   4   5   5   20   0  16  17
 2  2    5   4   5   4   5   5   5   5   5   4   5   5   5   5   5   4   5   4   4   5   20   0  16  17
 2  2    5   5   5   4   5   5   5   4   5   5   5   5   5   5   4   5   4   4   5   5   20   0  16  17
 2  2    5   5   5   4   5   5   5   5   5   5   5   5   5   5   4   5   4   4   5   5   20   0  16  17
 2  2    5   5   5   4   5   5   5   5   5   5   5   5   5   5   5   5   4   4   5   5   20   0  16  17
 2  2    5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   4   5   5   5   20   0  16  17
 2  2    5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   20   0  16  17
 2  0    5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   20   0  16  17
 2  0    5   5   5   5   5   5   5   5   5   5   5   5   5   5   6   5   5   5   5   5   20   1  16  17
 2  0    5   5   5   6   5   5   5   5   5   5   5   5   5   5   5   6   5   5   5   5   20   2  16  17
 2  0    5   5   5   6   5   5   5   5   5   5   5   5   5   5   6   6   5   5   5   5   20   3  16  17
 2  0    5   5   5   6   5   5   5   6   5   5   5   5   5   5   6   6   5   5   5   5   20   4  16  17
 2  0    5   6   5   6   5   5   5   5   6   5   5   5   5   5   6   6   5   5   5   5   20   5  16  17
 2  0    5   6   5   6   5   5   5   6   6   5   5   5   5   6   5   6   6   5   5   5   20   6  16  17
 2  0    5   6   5   6   5   5   5   6   6   5   5   5   5   6   5   6   6   5   5   5   20   7  16  17
 2  0    6   6   5   6   5   5   6   6   6   5   5   5   5   6   5   6   6   5   5   5   20   8  16  17
 2  0    6   6   5   6   5   5   6   6   6   5   5   5   5   6   5   6   6   6   5   5   20   9  16  17
 2  0    6   6   5   6   5   5   6   6   6   5   5   5   6   5   6   5   6   6   6   5   20  10  16  17
 2  0    6   6   5   6   6   5   6   6   6   5   5   6   5   6   5   6   6   6   5   5   20  11  16  17
 2  0    6   6   5   6   6   5   6   6   6   5   5   6   5   6   5   6   6   6   5   5   20  12  16  17
 2  0    6   6   5   6   6   5   6   6   6   6   6   6   5   6   5   6   6   6   5   5   20  13  16  17
 2  0    6   6   6   6   6   5   6   6   6   6   6   6   5   6   5   6   6   6   5   5   20  14  16  17
 2  0    6   6   6   6   6   5   6   6   6   6   6   6   5   6   5   6   6   6   5   5   20  15  16  17
 2  0    6   6   6   6   6   5   6   6   6   6   6   6   5   6   6   6   6   6   5   5   20  16  16  17
 2  0    6   6   6   6   6   5   6   6   6   6   6   6   5   6   6   6   6   6   5   6   20  17  16  17
 2  0    6   6   6   6   6   5   6   6   6   6   6   6   5   6   6   6   6   6   6   6   20  18  16  17
 2  0    6   6   6   6   6   5   6   6   6   6   6   6   6   6   6   6   6   6   6   6   20  19  16  17
 2  0    6   6   6   6   6   6   6   6   6   6   6   6   6   6   6   6   6   6   6   6   20  20  16  17
 2  0    6   6   6   6   6   6   6   6   6   6   6   6   6   6   6   8   6   6   6   6   20  20  16  17
 2  0    6   6   6   6   6   6   6   6   8   6   6   6   6   6   6   8   6   6   6   6   20  20  16  17
 2  0    6   6   6   8   6   6   6   6   8   6   6   6   6   6   6   8   6   6   6   6   20  20  16  17
 2  0    6   6   6   8   6   6   6   6   8   6   6   6   6   6   6   8   6   6   6   6   20  20  16  17
 2  0    6   6   6   8   6   6   6   6   8   6   6   6   6   6   6   8   6   6   6   6   20  20  16  17
 2  0    6   8   6   8   6   6   6   6   8   6   6   6   6   8   6   8   8   6   6   6   20  20  16  17
 2  0    6   8   6   8   6   6   6   6   8   6   6   6   6   8   6   8   8   6   6   6   20  20  16  17
 2  0    6   8   6   8   6   6   6   8   8   6   6   6   6   8   6   8   8   6   6   6   20  20  16  17
 2  0    8   8   6   8   6   6   6   8   8   6   6   6   6   8   6   8   8   6   6   6   20  20  16  17
 2  0    8   8   6   8   6   6   8   8   8   6   6   6   6   8   6   8   8   8   6   6   20  20  16  17
 2  0    8   8   6   8   6   6   8   8   8   6   6   6   8   8   6   8   8   8   6   6   20  20  16  17
 2  0    8   8   6   8   8   6   8   8   8   6   6   8   6   8   6   8   8   8   6   6   20  20  16  17
 2  0    8   8   6   8   8   6   8   8   8   8   8   8   6   8   6   8   8   8   6   6   20  20  16  17
 2  0    8   8   6   8   8   6   8   8   8   8   8   8   6   8   6   8   8   8   6   6   20  20  16  17
 2  0    8   8   8   8   8   6   8   8   8   8   8   8   6   8   6   8   8   8   6   6   20  20  16  17
 2  0    8   8   8   8   8   6   8   8   8   8   8   8   6   8   6   8   8   8   6   6   20  20  16  17
 2  0    8   8   8   8   8   6   8   8   8   8   8   8   6   8   8   8   8   8   6   6   20  20  16  17
 2  0    8   8   8   8   8   6   8   8   8   8   8   8   6   8   8   8   8   8   6   8   20  20  16  17
 2  0    8   8   8   8   8   6   8   8   8   8   8   8   6   8   8   8   8   8   6   8   20  20  16  17
 2  0    8   8   8   8   8   6   8   8   8   8   8   8   6   8   8   8   8   8   7   8   20  20  16  17
 2  0    8   8   8   8   8   6   8   8   8   8   8   8   8   8   8   8   8   8   7   8   20  20  16  17
 2  3    8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   7   8   20  20  16  17
 2  3    8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   20  20  16  17
ara@ferrari:~/SO/Projeto2/semaphore_restaurant/run$
```

Figura 16 - Solução Disponibilizada



```
Run n.º 1
                                Restaurant - Description of the internal state

CH WT  C00 C01 C02 C03 C04 C05 C06 C07 C08 C09 C10 C11 C12 C13 C14 C15 C16 C17 C18 C19  ATT FIE 1st las
 0  0    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1    0   0   0  -1
 0  0    1   1   1   1   1   1   1   2   1   1   1   1   1   1   1   1   1   1   1   1    1   0   7  -1
 0  0    1   1   1   1   1   1   1   2   1   1   2   1   1   1   1   1   1   1   1   1    2   0   7  -1
 0  0    1   1   1   1   1   1   1   2   1   1   2   1   1   1   1   1   2   1   1   1    3   0   7  -1
 0  0    1   1   1   1   1   1   2   2   1   1   2   1   1   1   1   1   1   2   1   1    4   0   7  -1
 0  0    1   1   1   1   1   1   2   2   1   1   2   1   1   2   1   1   1   2   1   1    5   0   7  -1
 0  0    1   1   2   1   1   1   2   2   1   1   2   1   1   2   1   1   1   2   1   1    6   0   7  -1
 0  0    1   1   2   1   1   1   2   2   1   2   2   1   1   2   1   1   1   2   1   1    7   0   7  -1
 0  0    1   1   2   1   1   2   2   2   1   2   2   1   1   2   1   1   1   2   1   1    8   0   7  -1
 0  0    1   1   2   1   1   2   2   2   1   2   2   1   2   2   1   1   1   2   1   1    9   0   7  -1
 0  0    1   1   2   1   1   2   2   2   2   2   2   1   2   2   1   1   1   2   1   1   10   0   7  -1
 0  0    2   1   2   1   1   2   2   2   2   2   2   1   2   2   1   1   1   2   1   1   11   0   7  -1
 0  0    2   1   2   1   1   2   2   2   2   2   2   1   2   2   2   1   1   2   1   1   12   0   7  -1
 0  0    2   1   2   2   1   2   2   2   2   2   2   1   2   2   2   1   1   2   1   1   13   0   7  -1
 0  0    2   1   2   2   1   2   2   2   2   2   2   1   2   2   2   1   2   2   1   1   14   0   7  -1
 0  0    2   1   2   2   1   2   2   2   2   2   2   1   2   2   2   1   2   2   1   2   15   0   7  -1
 0  0    2   2   2   2   1   2   2   2   2   2   2   1   2   2   2   1   2   2   1   2   16   0   7  -1
 0  0    2   2   2   2   1   2   2   2   2   2   2   1   2   2   2   2   2   2   1   2   17   0   7  -1
 0  0    2   2   2   2   2   2   2   2   2   2   2   1   2   2   2   2   2   2   1   2   18   0   7  -1
 0  0    2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   1   2   19   0   7  -1
 0  0    2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   4   2   20   0   7  18
 0  0    2   2   2   2   2   2   2   3   2   2   2   2   2   2   2   2   2   2   4   2   20   0   7  18
 0  0    2   2   2   2   2   2   2   3   2   2   4   2   2   2   2   2   2   2   4   2   20   0   7  18
 0  0    2   2   2   2   2   2   2   3   2   2   4   2   2   2   2   2   2   4   4   2   20   0   7  18
 0  0    2   2   2   2   2   2   4   3   2   2   4   2   2   2   2   2   2   4   4   2   20   0   7  18
 0  0    2   2   4   2   2   2   4   3   2   2   4   2   2   2   2   2   2   4   4   2   20   0   7  18
 0  0    2   2   4   2   2   2   4   3   2   2   4   2   2   4   2   2   2   4   4   2   20   0   7  18
 0  0    2   2   4   2   2   2   4   3   2   4   4   2   2   4   2   2   2   4   4   2   20   0   7  18
 0  0    2   2   4   2   2   2   4   3   2   4   4   2   4   4   2   2   2   4   4   2   20   0   7  18
 0  0    2   2   4   2   2   2   4   3   4   4   4   2   4   4   2   2   2   4   4   2   20   0   7  18
 0  0    4   2   4   2   2   2   4   3   4   4   4   2   4   4   2   2   2   4   4   2   20   0   7  18
 0  0    4   2   4   2   2   4   4   3   4   4   4   2   4   4   2   2   2   4   4   2   20   0   7  18
 0  0    4   2   4   2   2   4   4   3   4   4   4   2   4   4   4   2   2   4   4   2   20   0   7  18
 0  0    4   2   4   4   2   4   4   3   4   4   4   2   4   4   4   2   2   4   4   2   20   0   7  18
 0  0    4   2   4   4   2   4   4   3   4   4   4   2   4   4   4   2   4   4   4   2   20   0   7  18
 0  0    4   2   4   4   2   4   4   3   4   4   4   2   4   4   4   2   4   4   4   2   20   0   7  18
 0  0    4   2   4   4   2   4   4   3   4   4   4   2   4   4   4   2   4   4   4   4   20   0   7  18
 0  0    4   4   4   4   2   4   4   3   4   4   4   2   4   4   4   2   4   4   4   4   20   0   7  18
 0  0    4   4   4   4   2   4   4   3   4   4   4   2   4   4   4   4   4   4   4   4   20   0   7  18
 0  0    4   4   4   4   4   4   4   3   4   4   4   2   4   4   4   4   4   4   4   4   20   0   7  18
 0  0    4   4   4   4   4   4   4   3   4   4   4   4   4   4   4   4   4   4   4   4   20   0   7  18
 0  0    4   4   4   4   4   4   4   3   4   4   4   4   4   4   4   4   4   4   4   4   20   0   7  18
 0  1    4   4   4   4   4   4   4   3   4   4   4   4   4   4   4   4   4   4   4   4   20   0   7  18
 0  1    4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0   7  18
 0  0    4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0   7  18
 1  0    4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0   7  18
 1  0    4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0   7  18
 2  0    4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0   7  18
 2  2    4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0   7  18
 2  2    4   4   4   4   4   4   4   4   4   4   5   4   4   4   4   4   4   4   4   4   20   0   7  18
 2  2    4   4   4   4   4   4   4   4   4   4   5   4   4   4   4   4   4   5   4   4   20   0   7  18
 2  2    4   4   5   4   4   4   4   5   4   4   4   5   4   4   4   4   4   4   5   4   20   0   7  18
 2  2    4   4   5   4   4   4   4   5   4   4   4   5   4   4   5   4   4   4   5   4   20   0   7  18
 2  2    4   4   5   4   4   4   4   5   4   4   5   5   4   4   5   4   4   4   5   4   20   0   7  18
 2  2    4   4   5   4   4   4   4   5   4   4   5   5   4   5   5   4   4   4   5   4   20   0   7  18
 2  2    5   4   5   4   4   4   4   5   4   4   5   5   5   4   5   5   4   4   5   4   20   0   7  18
 2  2    5   4   5   4   4   4   5   5   4   5   5   5   4   5   5   4   4   4   5   4   20   0   7  18
 2  2    5   4   5   4   4   4   5   5   4   5   5   5   4   5   5   5   4   4   5   4   20   0   7  18
 2  2    5   4   5   5   4   5   5   4   5   5   5   4   5   5   5   4   4   5   4   4   20   0   7  18
 2  2    5   4   5   5   4   5   5   4   5   5   5   4   5   5   5   4   5   5   4   4   20   0   7  18
 2  2    5   4   5   5   4   5   5   4   5   5   5   4   5   5   5   4   5   5   4   5   20   0   7  18
 2  2    5   5   5   5   4   5   5   4   5   5   5   4   5   5   5   4   5   5   4   5   20   0   7  18
 2  2    5   5   5   5   4   5   5   4   5   5   5   4   5   5   5   5   5   5   4   5   20   0   7  18
 2  2    5   5   5   5   5   5   5   4   5   5   5   4   5   5   5   5   5   5   4   5   20   0   7  18
 2  2    5   5   5   5   5   5   5   4   5   5   5   5   5   5   5   5   5   5   4   5   20   0   7  18
 2  2    5   5   5   5   5   5   5   4   5   5   5   5   5   5   5   5   5   5   5   5   20   0   7  18
 2  0    5   5   5   5   5   5   5   4   5   5   5   5   5   5   5   5   5   5   5   5   20   0   7  18
 2  0    5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   20   0   7  18
 2  0    5   5   5   5   5   5   5   5   5   5   6   5   5   5   5   5   5   5   5   5   20   0   7  18
 2  0    5   5   6   5   5   5   5   5   5   5   6   5   5   5   5   5   5   5   5   5   20   1   7  18
 2  0    5   5   6   5   5   5   5   5   5   5   6   5   5   5   6   5   5   5   5   5   20   2   7  18
 2  0    5   5   6   5   5   5   5   5   6   5   6   5   5   5   6   5   5   5   5   5   20   3   7  18
 2  0    5   5   6   5   5   5   5   6   5   5   6   5   5   6   6   5   5   5   5   5   20   4   7  18
 2  0    5   5   6   5   5   5   5   5   6   5   6   5   6   6   6   5   5   5   5   5   20   5   7  18
 2  0    5   5   6   5   5   5   5   5   6   5   6   6   6   6   6   5   5   5   5   5   20   6   7  18
 2  0    5   5   6   5   6   5   5   5   6   5   6   6   6   6   6   5   5   5   5   5   20   7   7  18
 2  0    5   5   6   5   6   5   5   5   6   5   6   6   6   6   6   5   6   5   5   5   20   8   7  18
 2  0    5   5   6   6   6   5   5   5   6   5   6   6   6   6   6   5   6   5   5   5   20   9   7  18
 2  0    5   5   6   6   6   5   5   5   6   5   6   6   6   6   6   6   6   5   5   5   20  10   7  18
 2  0    5   5   6   6   6   5   5   5   6   6   6   6   6   6   6   6   6   5   5   5   20  11   7  18
 2  0    5   5   6   6   6   5   5   5   6   6   6   6   6   6   6   6   6   5   5   6   20  12   7  18
 2  0    5   6   6   6   6   5   5   5   6   6   6   6   6   6   6   6   6   5   5   6   20  13   7  18
 2  0    5   6   6   6   6   5   5   5   6   6   6   6   6   6   6   6   6   5   5   6   20  14   7  18
 2  0    5   6   6   6   6   5   5   6   6   6   6   6   6   6   6   6   6   5   5   6   20  15   7  18
 2  0    6   6   6   6   6   5   5   6   6   6   6   6   6   6   6   6   6   5   5   6   20  16   7  18
 2  0    6   6   6   6   6   5   6   6   6   6   6   6   6   6   6   6   6   5   6   6   20  17   7  18
 2  0    6   6   6   6   6   6   6   6   6   6   6   6   6   6   6   6   6   5   6   6   20  18   7  18
 2  0    6   6   6   6   6   6   6   6   6   6   6   6   6   6   6   6   6   6   6   6   20  19   7  18
 2  0    6   6   6   6   6   6   6   6   6   6   8   6   6   6   6   6   6   6   6   6   20  20   7  18
 2  0    6   6   8   6   6   6   6   6   6   6   8   6   6   6   6   6   6   6   6   6   20  20   7  18
 2  0    6   6   8   6   6   6   6   6   6   8   8   6   6   6   6   6   6   6   6   6   20  20   7  18
 2  0    6   6   8   6   6   6   6   6   8   6   8   6   6   6   8   6   6   6   6   6   20  20   7  18
 2  0    6   6   8   6   6   6   6   6   8   6   8   8   6   6   8   6   6   6   6   6   20  20   7  18
 2  0    6   6   8   6   6   6   6   6   8   6   8   8   8   8   8   6   6   6   6   6   20  20   7  18
 2  0    6   6   8   6   8   6   6   6   8   6   8   8   8   8   8   8   6   6   6   6   20  20   7  18
 2  0    6   6   8   6   8   6   6   6   8   6   8   8   8   8   8   6   8   6   6   6   20  20   7  18
 2  0    6   6   8   8   8   6   6   6   8   6   8   8   8   8   8   8   8   6   6   6   20  20   7  18
 2  0    6   6   8   8   8   6   6   6   8   8   8   8   8   8   8   8   8   6   6   6   20  20   7  18
 2  0    6   8   8   8   8   6   6   6   8   8   8   8   8   8   8   8   8   6   6   6   20  20   7  18
 2  0    6   8   8   8   8   8   6   6   8   8   8   8   8   8   8   8   8   6   6   8   20  20   7  18
 2  0    6   8   8   8   8   8   6   6   8   8   8   8   8   8   8   8   8   6   7   8   20  20   7  18
 2  0    6   8   8   8   8   8   6   6   8   8   8   8   8   8   8   8   8   6   7   8   20  20   7  18
 2  0    8   8   8   8   8   8   6   8   8   8   8   8   8   8   8   8   8   6   7   8   20  20   7  18
 2  0    8   8   8   8   8   8   6   8   8   8   8   8   8   8   8   8   8   6   8   7   20  20   7  18
 2  0    8   8   8   8   8   8   6   8   8   8   8   8   8   8   8   8   8   8   7   8   20  20   7  18
 2  3    8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   7   8   20  20   7  18
 2  3    8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   20  20   7  18
ara@ferrari:~/SO/Projeto2/semaphore_restaurant/run$
```

Figura 17 - Resultados Obtidos

5 - Conclusão

Concluindo, este programa permite visualizar todos os estados presentes de cada uma das entidades ao longo do tempo, estando todos os processos sincronizados e existindo uma dinâmica lógica entre eles, graças à implementação de semáforos.

Este trabalho, serviu para clarificar os nosso conhecimentos em diferentes aspetos tanto a nível de interpretação e utilização de semáforos em C como na escrita e interpretação da própria linguagem em C.

Neste ponto da realização do trabalho/relatório podemos afirmar que conseguimos alcançar todos os objetivos que o guião propunha, sendo desta forma muito satisfatória a realização do projeto.

Ao longo do projeto, o mesmo foi submetido a vários testes e guardado num repositório de GitHub para que pudéssemos estar sempre a par de erros e ultrapassá-los com mais facilidade.

6 - Bibliografia

Na realização deste trabalho foram consultados os slides teóricos bem como os guiões práticos disponibilizados no e-learning.

Para além destes, foram visitados alguns sites entre 18/12/2022 e 06/01/2023, tais como:

- https://stackoverflow.com/