Licenciatura em Engenharia Informática

Trabalho 2

Jantar de Amigos

Joana Gomes 104429
Lia Cardoso 107548
Turma P5

Percentagem de contribuição para o Trabalho: 50%/50%

06/01/2023

Índice

Introdução

No âmbito da Unidade Curricular de Sistemas Operativos foi-nos proposto o trabalho 2 intitulado “Jantar de Amigos (Restaurant)” cujo objetivo é a compreensão dos mecanismos associados à execução e sincronização de processos e threads. Neste trabalho pretende-se que se crie uma aplicação em C que simule o seguinte jantar:

“Um grupo de amigos combinou um jantar num restaurante que tem apenas um empregado de mesa (waiter) e um cozinheiro (chef). As regras são que o primeiro a chegar faz o pedido da comida, mas apenas depois de todos chegarem. O empregado de mesa deve levar o pedido ao cozinheiro e trazer a comida para a mesa quando esta estiver pronta. No final, os amigos apenas devem abandonar a mesa depois de todos terminarem de comer, sendo que o último a chegar ao restaurante tem a responsabilidade de pagar a conta.”

Para tal, tomou-se como ponto de partida o código fonte disponibilizado pelos docentes da unidade curricular e alterou-se as zonas assinaladas no mesmo.

Ao longo deste relatório é descrita a abordagem que foi utilizada para desenvolver a aplicação e os testes realizados para validar a solução.

Semáforos

De modo a desenvolver a aplicação desejada, foram utilizados semáforos. Os clientes, o empregado e o cozinheiro são processos independentes, sendo a sua sincronização realizada através dos semáforos e memória partilhada. Todos os processos são criados no início do programa e estão em execução a partir dessa altura. Os processos estão ativos apenas quando for necessário, devendo bloquear sempre que têm de esperar por algum evento.

Esses semáforos estão explicados na seguinte tabela:

Semáforo	Entidade Down	Função Down	# Downs	Entidade Up	Função Up	# Ups
friendsArrived	Clients	waitFriends()	TABLESIZE (20)	Clients (last)	waitFriends()	TABLESIZE (20)
requestReceived	caso 1: Clients (first) caso 2: Clients (last)	caso 1: orderFood() caso 2: waitAndPay()	2	Waiter	waitForClientOrChef()	2
foodArrived	Clients	waitFood()	TABLESIZE (20)	Waiter	takeFoodToTable()	TABLESIZE (20)
allFinished	Clients	waitAndPay()	TABLESIZE (20)	Clients	waitAndPay()	TABLESIZE (20)
waiterRequest	caso 1: Clients (first) caso 2: Clients (last) caso 3: Chef	caso 1: orderFood() caso 2: waitAndPay() caso 3: processOrder()	3	Waiter	waitForClientOrChef()	3
waitOrder	Chef	waitForOrder()	1	Waiter	informChef()	1

Figura 1 - Tabela de Semáforos

Código

Para desenvolver a aplicação adicionou-se código nas zonas assinaladas nos ficheiros pré-disponibilizados pelos docentes denominados ***semSharedMemClient.c***, ***semSharedMemWaiter.c*** e ***semSharedMemChef.c*** destinados aos clientes, empregado e cozinheiro, respetivamente.

Clientes

A primeira função onde foi adicionado código foi a ***waitFriends()*** no processo dos clientes (*semSharedMemClients.c*).

Inicialmente alterou-se o estado dos clientes para *WAIT_FOR_FRIENDS* e salvou-se essa alteração. Então, fez-se uma incrementação à variável *tableClients* que guarda o número de clientes que já chegaram ao restaurante. Dessa forma foi possível identificar o ID do primeiro cliente a chegar à mesa através de uma condição if (*if tableClients == 1*) e guardou-se o ID na variável *tableFirst*, assim como o do último cliente (*if tableClients == TABLESIZE*), - a variável *TABLESIZE* é o número total de clientes que irão ocupar a mesa - e guardou-se o ID em *tableLast*.

Todos os clientes que se sentam à mesa ficam à espera que o resto dos amigos cheguem, para isso, utilizou-se o semáforo ***friendsArrived***, ao qual todos os amigos fazem um *down* e ficam aí bloqueados até que, enfim, chega o último cliente que faz 20 *ups* ao mesmo semáforo para todos avançarem para a próxima função.

Por fim, a função vai retornar a variável *bool first*, sendo *true* caso se trate do primeiro cliente e *false* caso não seja.

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

    sh->fSt.tableClients++;
    if (sh->fSt.tableClients == 1){
        sh->fSt.tableFirst = id;
        first = true;
    }
    if (sh->fSt.tableClients == TABLESIZE){
        sh->fSt.tableLast = id;
        for (int i=0; i<=20; i++){
            if (semUp (semgid, sh->friendsArrived) == -1)
            { perror ("error on the up operation for semaphore access (CT)");
                exit (EXIT_FAILURE);
            }
        }

    }


    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->friendsArrived) == -1)
    { perror ("error on the down operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
    }

    return first;
}
```

Figura 2 - Função waitFriends()

Após isto, o primeiro cliente vai para a função ***orderFood()*** que começa por alterar o seu estado para *FOOD_REQUEST* e guardá-lo. Faz também uma incrementação de um valor à variável *foodRequest* que vai servir de *flag* para o empregado.

De seguida é feito um *up* ao semáforo ***waiterRequest*** que até aqui estava a bloquear o empregado para este receber um pedido, neste caso, do primeiro cliente.

Por fim, faz-se um *down* ao semáforo ***requestReceived*** para o cliente esperar que o empregado receba o pedido antes de poder prosseguir.

```
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.st.clientStat[id] = FOOD_REQUEST;
    saveState(nFic,&sh->fSt);

    sh->fSt.foodRequest++;

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


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

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

Figura 3 - Função orderFood()

Na função ***waitFood()*** altera-se o estado dos clientes para *WAIT_FOR_FOOD*.

De maneira a que os clientes esperem que a comida chegue foi feito 20 *downs* (um por cada cliente) ao semáforo ***foodArrived*** ao qual o empregado dará 20 *ups* após trazer a comida para a mesa, dando permissão aos clientes para avançarem e alterarem o seu estado para *EAT*.

```
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 */
    //clients ESPERAM até a comida chegar
    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 4 - Função waitFood()

Finalmente, os clientes avançam para a última função ***waitAndPay()*** e vão incrementando a variável *tableFinishEat* e mudam o seu estado para *WAIT_FOR_OTHERS*.

Todos os clientes exceto o último cliente a terminar de comer fazem *down* ao semáforo ***allFinished*** para que esperem por todos acabarem, já o último (*tableFinishEat* == *TABLESIZE*) dá 20 *ups* a este semáforo para o desbloquear para todos.

Também se verifica se se trata do último cliente. Através de uma condição *if* compara-se o ID dos clientes com o ID do último cliente guardado anteriormente em *tableLast*. alterando a variável *bool last* para *true*.

```
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 */
    sh->fSt.tableFinishEat++;
    sh ->fSt.st.clientStat[id] = WAIT_FOR_OTHERS;
    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 (id == sh->fSt.tableLast) {last = true;}

    if (sh->fSt.tableFinishEat == TABLESIZE){
        for (int i=0; i<20; i++){
            if (semUp (semgid, sh->allFinished) == -1) {
                perror ("error on the up operation for semaphore access (CT)");
                exit (EXIT_FAILURE);
            }
        }
    }
    else{
        if (semDown (semgid, sh->allFinished) == -1) {
            perror ("error on the down operation for semaphore access (CT)");
            exit (EXIT_FAILURE);
        }
    }
```

Figura 5 - Parte 1 da função waitAndPay()

Caso se trate do último cliente, este muda o seu estado para *WAIT_FOR_BILL* depois de incrementar a variável *paymentRequest* que servirá de *flag* para o empregado.

O último cliente é responsável por pagar ao empregado, logo tem de dar up ao semáforo waiterRequest para o desbloquear e para esperar que o empregado receba o seu pedido de pagamento faz um down ao semáforo requestReceived cujo o empregado irá desbloquear.

Por fim, todos os clientes alteram o seu estado para FINISHED.

```
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.paymentRequest++;
    sh->fSt.st.clientStat[id] = WAIT_FOR_BILL;
    saveState(nFic, &sh->fSt);


    if (semUp (semgid, sh->waiterRequest) == -1) {
    perror ("error on the up 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 */
    //ESPERA que o waiter receba o pedido
    if (semDown (semgid, sh->requestReceived) == -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] = 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 6 - Parte 2 da função waitAndPay()

Empregado

A primeira função no processo do empregado (*semSharedMemWaiter.c*) intitulada de waitForClientOrChef() tem início com a inicialização da variável ret que servirá para avançar no período de vida do empregado:

```
int req, nReq=0;
while(nReq<3) {
    req = waitForClientOrChef();
    switch(req) {
        case FOODREQ:
                informChef();
                break;
        case FOODREADY:
                takeFoodToTable();
                break;
        case BILL:
                receivePayment();
                break;
    }
    nReq++;
```

Figura 7 - Variável ret

É realizada a alteração do estado do empregado para WAIT_FOR_REQUEST.

Após isto faz-se um down ao semáforo waiterRequest onde o empregado vai ficar bloqueado até ou um cliente ou o cozinheiro fizerem algum pedido.

```
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);
    }

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

Figura 8 - Parte 1 da função waitForClientOrChef()

Em seguida, é analisado que pedido foi feito. Se a variável *foodRequest* for igual a 1, ou seja, se o primeiro cliente a incrementou, é porque foi feito o pedido da comida ao empregado, este notifica o cliente que recebeu o pedido dando *up* ao semáforo ***requestReceived*** e então atualiza a variável *ret* para 1 (*FOODREQ*), que, como mostrado anteriormente vai conduzir o empregado à função ***informChef()***.

Se a variável *foodReady* for 1, é porque foi incrementada pelo cozinheiro para notificar o empregado que a comida está pronta e este deve levá-la aos clientes. Então *ret* é atualizado para 2 (*FOODREADY*) e a próxima função será a ***takeFoodToTable()***.

Por fim, se for a variável *paymentRequest* igual a 1, foi porque o último cliente a incrementou, o que significa que este está pronto para fazer o pagamento. O empregado notifica o cliente que recebeu este pedido e *ret* é igual a 3 (*BILL*), levando o empregado para a função *receivePayment()*.

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

    ret = 1;
}

if(sh->fSt.foodReady == 1){

    ret = 2;
}

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

    ret = 3;
}
if (semUp (semgid, sh->mutex) == -1) {
 perror ("error on the down operation for semaphore access (WT)");
    exit (EXIT_FAILURE);
}

return ret;
```

Figura 9 - Parte 2 da função waitForClientOrChef()

A função ***informChef()*** começa por atualizar o estado do empregado para *INFORM_CHEF* e incrementar a variável *foodOrder* que servirá de *flag* para o cozinheiro de modo a que este comece a cozinhar, para o mesmo efeito é dado um *up* ao semáforo ***waitOrder*** que até aqui estava a bloquear o cozinheiro.

```
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.st.waiterStat = INFORM_CHEF;
    saveState(nFic,&sh->fSt);

    sh->fSt.foodOrder++;

    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->waitOrder) == -1) {
        perror ("error on the up operation for semaphore access (PT)");
        exit (EXIT_FAILURE);
    }

}
```

Figura 10 - Função informChef()

A função ***takeFoodToTable()*** começa por decrementar a variável *foodReady* para não retornar *ret* igual a 2 (*FOODREADY*) novamente, e atualiza também o estado do empregado para *TAKE_TO_TABLE*.

Finalmente, o empregado dá *up* ao semáforo ***foodArrived*** 20 vezes para desbloquear todos os clientes que estavam à espera da sua refeição.

```
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.foodReady--;
    sh->fSt.st.waiterStat = TAKE_TO_TABLE;
    saveState(nFic,&sh->fSt);

    //WAITER desbloqueia o sem foodArrived para TODOS os clientes
    for (int i=0; i<20; i++){
        if (semUp (semgid, sh->foodArrived) == -1) {
        perror ("error on the up operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
        }

    }


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

Figura 11 - Função takeFoodToTable()

Por fim, a função receivePayment() atualiza o estado do empregado para RECEIVE_PAYMENT.

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

Figura 12 - Função receivePayment()

Cozinheiro

O processo do cozinheiro (*semSharedMemChef.c*) começa pela função ***waitForOrder()***, onde é feito um down ao semáforo ***waitOrder*** que será desbloqueado pelo empregado quando o cozinheiro for necessário. Nesta função o estado do cozinheiro é atualizado para *WAIT_FOR_ORDER*.

```
static void waitForOrder ()
{
    /* insert your code here */
    //waitOrder
    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.st.chefStat = WAIT_FOR_ORDER;
    saveState(nFic,&sh->fSt);

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

Figura 13 - Função waitForOrder()

Para terminar, a função processOrder() começa por verificar se foi feito um incremento à variável foodOrder, caso esta seja igual a 1 o cozinheiro atualiza o seu estado para COOK e, de modo a atualizar a informação para não haver repetições do período de vida do empregado, faz-se decrementos às variáveis foodRequest e foodOrder e um incremento a foodReady de modo a notificar o empregado que a comida está pronta a servir.

Após isto ser concluído, como foodOrder é diferente de 1, o estado do cozinheiro é REST.

O cozinheiro deve dar up ao semáforo waiterRequest após ter a comida pronta para desbloquear o empregado e este a levar à mesa.

```
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 */
    if(sh->fSt.foodOrder == 1){

        sh->fSt.st.chefStat = COOK;
        saveState(nFic,&sh->fSt);
        sh->fSt.foodRequest--;
        sh->fSt.foodOrder--;
        sh->fSt.foodReady++;
    }

    if(sh->fSt.foodOrder != 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 down operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }

}
```

Figura 14 - Função processOrder()

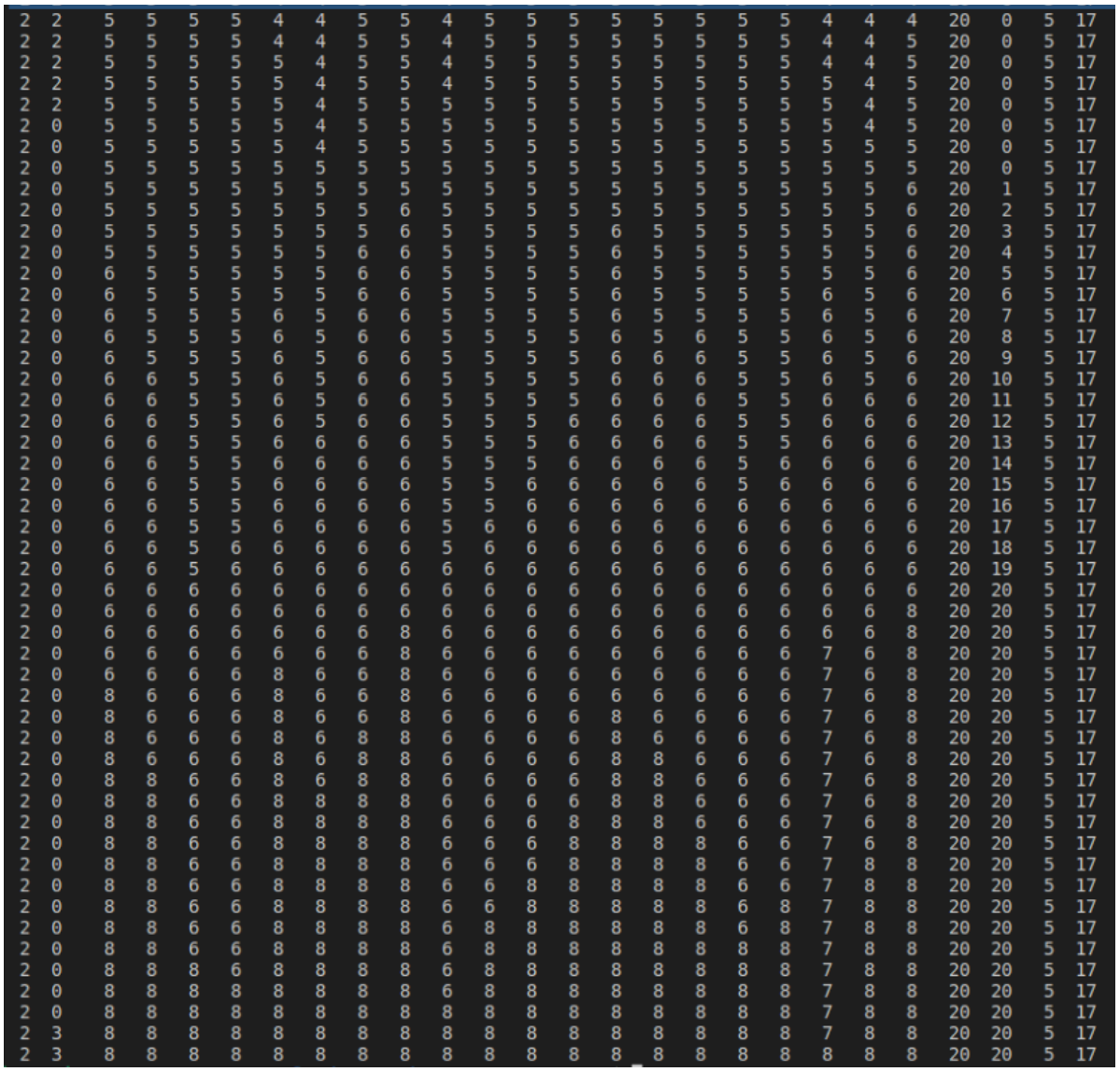
Resultado

Um exemplo de solução da aplicação é o seguinte:

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	2	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	2	1	1	1	1	1	1	2	1	1	1	1	1	1	1	1	0	5	-1
0	0	1	2	1	1	1	2	1	1	1	1	1	1	2	1	1	1	1	1	1	1	2	0	5	-1
0	0	1	2	1	1	1	2	1	1	1	1	1	1	2	1	1	2	1	1	1	1	3	0	5	-1
0	0	1	2	1	1	1	2	1	1	1	1	1	2	2	1	1	2	1	1	1	1	4	0	5	-1
0	0	2	2	1	1	1	2	1	1	1	1	1	2	2	1	1	2	1	1	1	1	5	0	5	-1
0	0	2	2	1	1	1	2	1	2	1	1	1	2	2	1	1	2	1	1	1	1	6	0	5	-1
0	0	2	2	1	1	1	2	1	2	1	1	1	2	2	1	2	2	1	1	1	1	7	0	5	-1
0	0	2	2	1	2	1	2	1	2	1	1	1	2	2	1	2	2	1	1	1	1	8	0	5	-1
0	0	2	2	1	2	1	2	1	2	1	1	2	2	2	1	2	2	1	1	1	1	9	0	5	-1
0	0	2	2	1	2	1	2	2	2	1	1	2	2	2	1	2	2	1	1	1	1	10	0	5	-1
0	0	2	2	1	2	1	2	2	2	1	1	2	2	2	2	2	2	1	1	1	1	11	0	5	-1
0	0	2	2	2	2	1	2	2	2	1	1	2	2	2	2	2	2	1	1	1	1	12	0	5	-1
0	0	2	2	2	2	1	2	2	2	1	2	2	2	2	2	2	2	1	1	1	1	13	0	5	-1
0	0	2	2	2	2	1	2	2	2	1	2	2	2	2	2	2	2	2	1	1	1	14	0	5	-1
0	0	2	2	2	2	1	2	2	2	1	2	2	2	2	2	2	2	2	1	1	2	15	0	5	-1
0	0	2	2	2	2	1	2	2	2	2	2	2	2	2	2	2	2	2	1	1	2	16	0	5	-1
0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	2	17	0	5	-1
0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	2	2	18	0	5	-1
0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	19	0	5	-1
0	0	2	2	2	2	2	2	2	2	2	2	2	2	4	2	2	2	2	2	2	2	20	0	5	17
0	0	2	2	2	2	2	2	2	2	2	2	2	4	4	2	2	2	2	2	2	2	20	0	5	17
0	0	2	2	2	2	2	3	2	2	2	2	2	4	4	2	2	2	2	2	2	2	20	0	5	17
0	0	2	4	2	2	2	3	2	2	2	2	2	4	4	2	2	2	2	2	2	2	20	0	5	17
0	0	4	4	2	2	2	3	2	2	2	2	2	4	4	2	2	2	2	2	2	2	20	0	5	17
0	0	4	4	2	2	2	3	2	2	2	2	2	4	4	2	2	4	2	2	2	2	20	0	5	17
0	0	4	4	2	2	2	3	2	4	2	2	2	4	4	2	2	4	2	2	2	2	20	0	5	17
0	0	4	4	2	2	2	3	2	4	2	2	2	4	4	2	4	4	2	2	2	2	20	0	5	17
0	0	4	4	2	4	2	3	2	4	2	2	2	4	4	2	4	4	2	2	2	2	20	0	5	17
0	0	4	4	2	4	2	3	2	4	2	2	4	4	4	2	4	4	2	2	2	2	20	0	5	17
0	0	4	4	2	4	2	3	4	4	2	2	4	4	4	2	4	4	2	2	2	2	20	0	5	17
0	0	4	4	2	4	2	3	4	4	2	2	4	4	4	4	4	4	2	2	2	2	20	0	5	17
0	0	4	4	4	4	2	3	4	4	2	2	4	4	4	4	4	4	2	2	2	2	20	0	5	17
0	0	4	4	4	4	2	3	4	4	2	4	4	4	4	4	4	4	2	2	2	2	20	0	5	17
0	0	4	4	4	4	2	3	4	4	2	4	4	4	4	4	4	4	2	2	4	2	20	0	5	17
0	0	4	4	4	4	2	3	4	4	4	4	4	4	4	4	4	4	4	2	4	2	20	0	5	17
0	0	4	4	4	4	4	3	4	4	4	4	4	4	4	4	4	4	4	4	4	4	20	0	5	17
0	1	4	4	4	4	4	3	4	4	4	4	4	4	4	4	4	4	4	4	4	4	20	0	5	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	5	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	5	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	5	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	5	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	5	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	5	17
2	2	4	4	4	4	4	4	4	4	4	4	4	5	4	4	4	4	4	4	4	4	20	0	5	17
2	2	4	5	4	4	4	4	4	4	4	4	4	5	4	4	4	4	4	4	4	4	20	0	5	17
2	2	4	5	4	4	4	4	4	4	4	4	4	5	4	4	4	5	4	4	4	4	20	0	5	17
2	2	4	5	4	4	4	4	4	5	4	4	4	5	4	4	4	5	4	4	4	4	20	0	5	17
2	2	4	5	4	4	4	4	4	5	4	4	4	5	5	4	4	5	4	4	4	4	20	0	5	17
2	2	5	5	4	4	4	4	4	5	4	4	4	5	5	4	4	5	4	4	4	4	20	0	5	17
2	2	5	5	4	4	4	4	4	5	4	4	4	5	5	4	5	5	4	4	4	4	20	0	5	17
2	2	5	5	4	5	4	4	4	5	4	4	4	5	5	4	5	5	4	4	4	4	20	0	5	17
2	2	5	5	4	5	4	4	5	5	4	4	5	5	5	4	5	5	4	4	4	4	20	0	5	17
2	2	5	5	4	5	4	4	5	5	4	4	5	5	5	5	5	5	4	4	4	4	20	0	5	17
2	2	5	5	5	5	4	4	5	5	4	4	5	5	5	5	5	5	4	4	4	4	20	0	5	17
2	2	5	5	5	5	4	4	5	5	4	5	5	5	5	5	5	5	4	4	4	4	20	0	5	17

Figura 15 - Parte 1 do exemplo de output



```
2 2 5 5 5 5 4 4 5 5 4 5 5 5 5 5 5 5 5 4 4 4 20 0 5 17
2 2 5 5 5 5 4 4 5 5 4 5 5 5 5 5 5 5 5 4 4 5 20 0 5 17
2 2 5 5 5 5 5 4 5 5 4 5 5 5 5 5 5 5 5 4 4 5 20 0 5 17
2 2 5 5 5 5 5 4 5 5 4 5 5 5 5 5 5 5 5 5 4 5 20 0 5 17
2 2 5 5 5 5 5 4 5 5 5 5 5 5 5 5 5 5 5 5 4 5 20 0 5 17
2 0 5 5 5 5 5 4 5 5 5 5 5 5 5 5 5 5 5 5 4 5 20 0 5 17
2 0 5 5 5 5 5 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 20 0 5 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 5 17
2 0 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 20 1 5 17
2 0 5 5 5 5 5 5 5 6 5 5 5 5 5 5 5 5 5 5 5 6 20 2 5 17
2 0 5 5 5 5 5 5 5 6 5 5 5 5 6 5 5 5 5 5 5 6 20 3 5 17
2 0 5 5 5 5 5 5 6 6 5 5 5 5 6 5 5 5 5 5 5 6 20 4 5 17
2 0 6 5 5 5 5 5 6 6 5 5 5 5 6 5 5 5 5 5 5 6 20 5 5 17
2 0 6 5 5 5 5 5 6 6 5 5 5 5 6 5 5 5 5 6 5 6 20 6 5 17
2 0 6 5 5 5 6 5 6 6 5 5 5 5 6 5 5 5 5 6 5 6 20 7 5 17
2 0 6 5 5 5 6 5 6 6 5 5 5 5 6 5 6 5 5 6 5 6 20 8 5 17
2 0 6 5 5 5 6 5 6 6 5 5 5 5 6 6 6 5 5 6 5 6 20 9 5 17
2 0 6 6 5 5 6 5 6 6 5 5 5 5 6 6 6 5 5 6 5 6 20 10 5 17
2 0 6 6 5 5 6 5 6 6 5 5 5 5 6 6 6 5 5 6 6 6 20 11 5 17
2 0 6 6 5 5 6 5 6 6 5 5 5 6 6 6 6 5 5 6 6 6 20 12 5 17
2 0 6 6 5 5 6 6 6 6 5 5 5 6 6 6 6 5 5 6 6 6 20 13 5 17
2 0 6 6 5 5 6 6 6 6 5 5 5 6 6 6 6 5 6 6 6 6 20 14 5 17
2 0 6 6 5 5 6 6 6 6 5 5 6 6 6 6 6 5 6 6 6 6 20 15 5 17
2 0 6 6 5 5 6 6 6 6 5 5 6 6 6 6 6 6 6 6 6 6 20 16 5 17
2 0 6 6 5 5 6 6 6 6 5 6 6 6 6 6 6 6 6 6 6 6 20 17 5 17
2 0 6 6 5 6 6 6 6 6 5 6 6 6 6 6 6 6 6 6 6 6 20 18 5 17
2 0 6 6 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 20 19 5 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 5 17
2 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 8 20 20 5 17
2 0 6 6 6 6 6 6 6 8 6 6 6 6 6 6 6 6 6 6 6 8 20 20 5 17
2 0 6 6 6 6 6 6 6 8 6 6 6 6 6 6 6 6 6 7 6 8 20 20 5 17
2 0 6 6 6 6 8 6 6 8 6 6 6 6 6 6 6 6 6 7 6 8 20 20 5 17
2 0 8 6 6 6 8 6 6 8 6 6 6 6 6 6 6 6 6 7 6 8 20 20 5 17
2 0 8 6 6 6 8 6 6 8 6 6 6 6 8 6 6 6 6 7 6 8 20 20 5 17
2 0 8 6 6 6 8 6 8 8 6 6 6 6 8 6 6 6 6 7 6 8 20 20 5 17
2 0 8 6 6 6 8 6 8 8 6 6 6 6 8 8 6 6 6 7 6 8 20 20 5 17
2 0 8 8 6 6 8 6 8 8 6 6 6 6 8 8 6 6 6 7 6 8 20 20 5 17
2 0 8 8 6 6 8 8 8 8 6 6 6 6 8 8 6 6 6 7 6 8 20 20 5 17
2 0 8 8 6 6 8 8 8 8 6 6 6 8 8 8 6 6 6 7 6 8 20 20 5 17
2 0 8 8 6 6 8 8 8 8 6 6 6 8 8 8 8 6 6 7 6 8 20 20 5 17
2 0 8 8 6 6 8 8 8 8 6 6 6 8 8 8 8 6 6 7 8 8 20 20 5 17
2 0 8 8 6 6 8 8 8 8 6 6 8 8 8 8 8 6 6 7 8 8 20 20 5 17
2 0 8 8 6 6 8 8 8 8 6 6 8 8 8 8 8 6 8 7 8 8 20 20 5 17
2 0 8 8 6 6 8 8 8 8 6 8 8 8 8 8 8 8 8 7 8 8 20 20 5 17
2 0 8 8 8 6 8 8 8 8 6 8 8 8 8 8 8 8 8 7 8 8 20 20 5 17
2 0 8 8 8 8 8 8 8 8 6 8 8 8 8 8 8 8 8 7 8 8 20 20 5 17
2 0 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 7 8 8 20 20 5 17
2 3 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 7 8 8 20 20 5 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 5 17
```

Figura 16 - Parte 2 do exemplo de output

Conclusão

Dado por terminada a realização da aplicação pedida e analisados os seus resultados, encontramo-nos satisfeitas com a sua resolução.

Ao longo deste trabalho pudemos alargar os nossos conhecimentos sobre a programação na linguagem C, sobre semáforos, ou seja, mecanismos associados à execução e sincronização de processos e threads, assim como pudemos melhorar as nossas capacidades de trabalho em grupo.

Bibliografia

No desenvolvimento do segundo trabalho prático de Sistemas Operativos "Jantar de Amigos - Restaurant" foram utilizados os seguintes recursos:

- Informação das aulas teórico-práticas disponibilizada na plataforma e-learning da Universidade de Aveiro na Unidade Curricular Sistemas Operativos;
- https://stackoverflow.com/ ;
- https://www.geeksforgeeks.org/ ;