2022/2023 – Sistemas Operativos

(Trabalho 02)

Jantar de Amigos – Restaurante

Trabalho realizado por:

Maria João Machado Sardinha (108756) – Turma P03

Rodrigo Martins Graça (107634) – Turma P01

Índice

Introdução 3

Comportamento dos Semáforos 4

Entidades - Clients 5

- Função waitFriends(): 5

- Função orderFood(): 7

- Função waitFood(): 8

- Função waitAndPay(): 10

Entidades - Waiter 13

- Função waitForClientsOrChef(): 13

- Função informChef(): 15

- Função takeFoodToTable(): 16

- Função receivePayment(): 17

Entidades - Chef 18

- Função waitForOrder(): 18

- Função processOrder(): 19

Testes efetuados para Validar os Resultados Obtidos 21

Conclusão 26

Fontes 27

Introdução

No âmbito da cadeira de Sistemas Operativos é no proposto, tomando como ponto de partida um código fonte dado, desenvolvermos uma aplicação em C com o objetivo de simularmos um jantar entre amigos.

Este jantar tem de obedecer a certas regras, tais como:
- Os elementos deste jantar são o grupo de amigos, um empregado de mesa e um cozinheiro;
- O primeiro amigo a chegar efetua o pedido da comida (mas apenas quando todos chegarem);
- O último amigo a chegar paga a conta;
- Os amigos abandonam a mesa, no fim, depois de todos acabarem de comer;
- O tamanho da mesa está ajustado ao número de amigos;
- O empregado de mesa leva o pedido ao cozinheiro e traz a comida quando esta estiver pronta.

Para este trabalho tivemos ainda de ter em conta que existem três tipos de identidade (os clientes (*Client*), o empregado de mesa (*Waiter*) e o cozinheiro (*Chef*)). Os processos destes são independentes e a sua sincronização é realizada através de semáforos e memória partilhada.

Neste relatório será analisado todo o código implementado.

Primeiramente mostraremos uma tabela e, depois, procederemos à explicação das funções de cada identidade, assim como, do código implementado em cada uma e, por fim, procederemos à exposição dos resultados e à conclusão do trabalho.

Comportamento dos Semáforos

Seguidamente encontra-se uma tabela que serve de suporte para verificar o comportamento de cada entidade. Procedemos à sua construção para uma melhor compreensão do código, assim como dos "*ups*" e "*downs*" de cada semáforo.

Semáforo	Up			Down	
	Entidade	Quando	Quantidade	Entidade	Quando
Mutex	Client	waitFriends	1	Client	waitFriends
		orderFood	1		orderFood
		waitFood	2		waitFood
		waitAndPay	3		waitAndPay
	Waiter	waitForClientOrChef	2	Waiter	waitForClientOrChef
		informChef	1		informChef
		takeFoodToTable	1		takeFoodToTable
		receivePayment	1		receivePayment
	Chef	waitForOrder	1	Chef	waitForOrder
		processOrder	1		processOrder
friendsArrived	Client	waitFriends	1	Client	waitFriends
waiterRequest	Client	orderFood	1	Waiter	waitForClientOrChef
		waitAndPay	1		
	Waiter	processOrder	1		
allFinished	Client	waitAndPay	1	Client	waitAndPay
requestReceived	Waiter	informChef	1	Client	orderFood
		receivePayment	1		waitAndPay
foodArrived	Waiter	takeFoodToTable	1	Client	waitFood
waitOrder	Waiter	informChef	1	Chef	waitForOrder

Entidades - Clients

Relativamente a esta identidade, os clientes (*Clients*), que correspondem aos amigos do jantar, o código alterado (em "*semSharedMemClient.c*") foi nas seguintes funções:

- Função waitFriends():

```
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.st.clientStat[id] = WAIT_FOR_FRIENDS;
        sh->fSt.tableFirst = id;
        saveState(nFic, &sh->fSt);
    }
    else if(sh->fSt.tableClients == TABLESIZE){
        sh->fSt.st.clientStat[id] = WAIT_FOR_FOOD;
        sh->fSt.tableLast = id;
        saveState(nFic, &sh->fSt);
        for(int i = 0; i < TABLESIZE; i++){
            if(semUp(semgid, sh->friendsArrived) == -1){
                perror("error on the up 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);
    }

    /* 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;
}
```

Esta função tem como objetivo fazer com que, quando cada amigo chega ao restaurante, este espere que os outros todos cheguem.

Para tal, o estado do número de clientes na mesa vai sendo atualizado, ou seja, conforme cada amigo chega, o valor de "`tableClients`" vai aumentando em 1.

```
sh->fSt.tableClients++;
```

De seguida é verificado qual o primeiro cliente a chegar através de uma condição "`if`". Esta condição verifica quando é que existe um amigo já na mesa e, depois, informa que o primeiro amigo já chegou (atribuindo o valor de verdade a "`first`"), atualiza o estado desse amigo para "`WAIT_FOR_FRIENDS`" (esperar pelos amigos), usando o id dele para o identificar. É ainda guardada a informação de que o seu id corresponde ao primeiro cliente a chegar (útil posteriormente para saber qual cliente vai efetuar o pedido da comida). Por fim, guarda-se o estado.

```
if(sh->fSt.tableClients == 1){
    first = true;
    sh->fSt.st.clientStat[id] = WAIT_FOR_FRIENDS;
    sh->fSt.tableFirst = id;
    saveState(nFic, &sh->fSt);
}
```

Caso não seja o primeiro cliente, ou seja, caso a condição acima não se verifique, vai-se verificar se já chegaram todos os amigos. Para tal, verifica-se se o número de clientes na mesa corresponde ao tamanho da mesa (visto que esta está ajustada ao número de amigos). Se esta condição se verificar, então já chegaram todos os amigos e o estado deste último a chegar é atualizado para "`WAIT_FOR_FOOD`" (esperar pela comida), usando o id dele para o identificar. É também guardada a informação de que o seu id corresponde ao último a chegar (o que é útil para depois se saber qual o amigo que vai pagar a conta). De seguida guarda-se o estado e realiza-se uma verificação relativamente ao estado de "`friendsArrived`".

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

Caso nenhuma destas condições seja verificada, então apenas atualizado o estado desse amigo para "`WAIT_FOR_FRIENDS`" (esperar pelos amigos), usando o id dele para o identificar e guarda-se o estado.

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

- Função orderFood():

```
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++;
    sh->fSt.st.clientStat[id] = FOOD_REQUEST;
    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 up 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);
    }
    ///
}
```

Esta função é o processo do cliente de fazer o pedido de comida ao empregado de mesa.

A função atualiza o número de pedidos de comida no estado atual do jantar, e atualiza o estado do primeiro cliente que é o que tem o id guardado para "FOOD_REQUEST" pois é este que vai fazer o pedido. E guarda-se o estado.

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

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

Em seguida, a função faz uma verificação do valor de "*semUp*" no semáforo "`waiterRequest`", que avisa o empregado de mesa de que há um pedido de comida.

Depois, a função faz a mesma verificação de valor de "*semUp*" no semáforo de *"mutex"* e na função "*semDown*" no semáforo "`requestReceived`", o que significa que o cliente vai bloquear até que o empregado de mesa indique que o pedido foi recebido. Nestas verificações, verifica-se se os valores estão corretos, ou seja, se estes forem iguais a -1 estão incorretos e, por consequência, é imprimida uma mensagem de erro e o programa termina.

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

- Função waitFood():

```
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 up 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);
    }
}
```

Esta função tem o objetivo de fazer o cliente esperar até que a comida esteja pronta.

A função começa por verificar o valor de "SemD*own*" do *mutex*, o que significa que passa a ter exclusivo acesso à memória partilhada.

Depois, a função atualiza o status do cliente para "WAIT_FOR_FOOD" (porque o cliente está à espera de comida).

Ela também chama a função "`saveState`", que salva o estado atual dos processos.

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

Em seguida, a função faz uma verificação do valor de "SemUp" no semáforo de *mutex*, saindo da região crítica.

De seguida, a função faz outra verificação para o valor de "SemDown" no semáforo "`foodArrived`", o que significa que o cliente vai bloquear até que a comida esteja pronta.

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

Depois de bloquear até que a comida esteja pronta, a função faz uma nova verificação de "SemDown" no semáforo de `mutex`, entrando de novo na região crítica. Isso é feito para garantir que apenas um processo acesse a memória partilhada de cada vez.

Depois, a função atualiza o status do cliente para "EAT". Ela também volta a guardar o estado atual dos processos.

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

Por último, a função faz uma verificação do valor de "SemUp" no semáforo de `mutex`, saindo da região crítica. Isso permite que outros processos acessem a memória partilhada.

- Função waitAndPay():

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

    if(sh->fSt.tableFinishEat == TABLESIZE){
        last = true;

        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);
    }
    ///

    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 */
        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);
```

Esta função tem como objetivo fazer com que todos os amigos esperem até que terminem todos a sua refeição e, após todos acabarem, faz com que o último amigo a chegar ao restaurante pague a conta.

Para que isto ocorra, começa-se por alterar o estado de cada cliente que já tenha acabado a sua refeição (usando o seu id para o identificar) para "`WAIT_FOR_FRIENDS`" (esperar pelos amigos). Após esta alteração, o número de amigos que já acabou de comer é atualizado, aumentando em um o valor de "`tableFinishEat`" e guarda-se o estado.

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

Caso o número de amigos que já acabaram de comer seja igual ao tamanho da mesa (ou seja, igual ao número de amigos), atribui-se à variável "`last`" o valor de verdade (`true`), pois esta variável no início da função foi iniciada como falso (false) e, no momento em que todos acabarem de comer, altera-se o seu valor para posteriormente se saber quando já se pode pedir a conta. Verifica-se ainda se o valor de "`semUp`" de "`allFinished`" está correto e, caso não esteja (se for igual a -1), aparece uma mensagem de erro e o programa termina.

```
if(sh->fSt.tableFinishEat == TABLESIZE){
    last = true;

    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);
        }
    }
}
```

Depois é ainda realizada uma verificação de se o valor de "semDown" de "allFinished" está correto e, caso não esteja (caso seja igual a -1), ocorre um erro e o programa imprime uma mensagem de erro e termina.

```
/* insert your code here */

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

Caso já todos tenham acabado de comer (se a variável "last" tiver valor "true"), então já podem pedir a conta.

Começa-se por aumentar o valor de "paymentRequest" somando-lhe 1 e muda-se o estado de "tableLast" para "WAIT_FOR_BILL" (esperar pela conta), para informar que o amigo que vai pagar já está à espera da conta pois já todos acabaram de comer.

Guarda-se o estado e faz-se uma verificação de se os valores de “semUp” de “waiterRequest”, de “semUp” de “mutex” e de “semDown” de “requestReceived” estão corretos, caso não estejam (se forem iguais a -1), aparece uma mensagem de erro e o programa termina.

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

Nesta função, para todos os clientes, conforme estes acabam de comer, é ainda atualizado o seu estado (identificando cada cliente pelo seu id) para “FINISHED” (finalizado) para indicar que estes já acabaram de comer e guarda-se o seu estado.

```
/* insert your code here */

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

Entidades - Waiter

Relativamente a esta identidade, o empregado de mesa (`Waiter`), que corresponde ao único empregado de mesa existente que vai servir o grupo de amigos no jantar, o código alterado (em "`semSharedMemWaiter.c`") foi nas seguintes funções:

- Função waitForClientsOrChef():

```c
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 */
    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 != 0){
        ret = FOODREQ;
        sh->fSt.foodRequest = 0;
    }
    else if(sh->fSt.foodReady != 0){
        ret = FOODREADY;
        sh->fSt.foodReady = 0;
    }
    else if(sh->fSt.paymentRequest != 0){
        sh->fSt.paymentRequest = 0;
        ret = BILL;
    }

    saveState(nFic, &sh->fSt);
    ///

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

    return ret;
}
```

Esta função tem como objetivo informar quando é que o empregado de mesa se encontra à espera quer do grupo de amigos, quer do chefe que lhes está a preparar a comida.

Para tal, quando chamada, começa-se por alterar o estado do empregado de mesa (*waiter*) para "WAIT_FOR_REQUEST" (à espera de pedido) para informar que o empregado de mesa se encontra à espera de ser chamado e guarda-se o seu estado.

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

Depois verifica-se se os valores de "semUp" e "semDown" de "mutex" e de "semDown" de "waiterRequest" estão corretos e, caso não estejam (caso correspondam a -1), aparece uma mensagem de erro e o programa termina.

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

/* 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);
}
```

Depois são realizadas algumas verificações (tendo em conta que a função irá devolver o valor de "ret":

- Caso o valor de "foodRequest" (pedido de comida) seja diferente de zero, altera-se o seu valor para zero e atribui-se à variável "ret" o valor "FOODREQ";
- Caso o ponto acima não se verifique e caso o valor de "foodReady" (comida pronta) seja diferente de zero, altera-se o seu valor para zero e atribui-se à variável "ret" o valor "FOODREADY";
- Caso os pontos acima não se verifiquem e caso o valor de "paymentRequest" (pedido de pagamento/conta) seja diferente de zero, altera-se o seu valor para zero e atribui-se à variável "ret" o valor "BILL";

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

saveState(nFic, &sh->fSt);
```

Deste modo, conseguimos sempre saber do que o empregado de mesa está à espera e mudar os seus estados. Por fim, salvam-se os estados.

- 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++;
    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);
    }
    ///
}
```

Esta função tem como objetivo o empregado de mesa ir informar o chef de que os clientes já realizaram o seu pedido de comida.

Começa-se por aumentar o valor de "`foodOrder`" (pedido de comida), somando-lhe 1 e por mudar o estado do empregado de mesa (*waiter*) para "`INFORM_CHEF`" (informar o chefe). Assim, temos a informação de que a comida já foi pedida e de que o empregado de mesa já foi informar o chefe de tal (para que este saiba que já pode começar a prepara a comida). Depois guardam-se os estados.

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

Nesta função é ainda verificado se os valores de "`semUp`" de "`mutex`", de "`semUp`" de "`requestReceived`" e de "`semUp`" de "`waitOrder`" estão corretos, caso não estejam (se corresponderem a -1), então é mostrada uma mensagem de erro e o programa termina.

```
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);
}
```

- Função takeFoodToTable():

Esta função serve para o momento em que o empregado de mesa leva a comida até à mesa do grupo de amigos.

```
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 < TABLESIZE; 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);
    }
}
```

Para começar, atualiza-se o estado do empregado de mesa (*waiter*) para “TAKE_TO_TABLE” (levar para a mesa) e salva-se o seu estado. Deste modo, sabemos que a comida já está a ser levada para a mesa.

Depois verifica-se se o valor de “SemUp” de “foodArrived” está correto e, caso não esteja (se este corresponder a -1), aparece uma mensagem de erro e o programa termina.

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

- Função receivePayment():

Esta função tem a finalidade de o empregado de mesa receber o pagamento do jantar.

```
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);
    }
}
```

A função faz o estado do empregado de mesa (*waiter*) atualizar para “RECEIVE_PAYMENT” (receber pagamento) e salvar o seu estado. Assim, sabemos que este já recolheu o pagamento feito pelo último cliente que chegou à mesa. Faz-se ainda uma verificação do valor do “semUp” de “requestReceived” (se este corresponder a -1 está incorreto e imprime uma mensagem de erro e termina o programa).

```
/* 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);
}
///
```

Entidades - Chef

Acerca desta identidade, o cozinheiro (Chef), que corresponde ao único cozinheiro existente que vai preparar a comida de todo o grupo de amigos no jantar, o código alterado (em "semSharedMemChef.c") foi nas seguintes funções:

- Função waitForOrder():

Esta função tem o objetivo de fazer o cozinheiro esperar, até ao momento em que o empregado de mesa lhe dá um pedido para começar a preparar.

```
static void waitForOrder ()
{
    /* insert your code here */
    if (semDown (semgid, sh->waitOrder) == -1) {
        perror ("error on the up 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);
    }
}
```

A função começa por verificar se o valor do "`semDown`" do "`waitOrder`" e, caso este esteja incorreto (se for igual a -1), imprime uma mensagem de erro e o programa termina.

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

A função altera o estado do cozinheiro para “COOK” quando este recebo o pedido do empregado de mesa. Assim como guarda esse estado.

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

- Função processOrder():

Esta função é o momento em que o cozinheiro está a preparar a comida para esta depois ser servida aos clientes.

```
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++;
    if(sh->fSt.foodReady != 0){
        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);
    }
    ///
}
```

Nesta função o cozinheiro já está a preparar os pedidos que lhe foram entregues pelo empregado de mesa e à medida que cada pedido é finalizado é adicionado 1 ao valor de "foodReady" (comida pronta).

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

Quando o valor de "foodReady" for diferente de 0, ou seja, quando o cozinheiro acabar de preparar todos os pedidos entregues pelo empregado de mesa, o cozinheiro passa a estar no estado "REST", e o estado é guardado.

```
if(sh->fSt.foodReady != 0){
    sh->fSt.st.chefStat = REST;
    saveState(nFic, &sh->fSt);
}
///
```

No fim desta função é ainda verificado se o valor de "semUp" de "waiterRequest" está correto, e, caso não esteja (caso corresponda a -1), o programa imprime uma mensagem de erro e termina.

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

Testes efetuados para Validar os Resultados Obtidos

De seguida apresentaremos alguns testes que efetuámos para validação dos resultados obtidos.

Primeiramente colocámos o programa a correr através do script "`run.sh`" fazendo, deste modo, o programa correr várias vezes. Visto que não ocorreu nenhum erro, chegámos à conclusão de que os resultados obtidos estão corretos.

De seguida, fomos verificar se os resultados que obtemos estavam corretos (se tinham lógica relativamente ao que era pedido no enunciado deste trabalho). Verificámos então todos os valores imprimidos, visualizando cada linha atentamente, e, visto que as atualizações que apareciam faziam sentido, chegámos novamente à conclusão de que estes resultados estariam corretos.

Com o objetivo de que este relatório não fique muito longo com vários prints dos resultados obtidos e visto que o código que levou à obtenção destes resultados estará a complementar este relatório, iremos apenas mostrar alguns prints destes resultados.

Os resultados encontram-se nas páginas que se seguem.

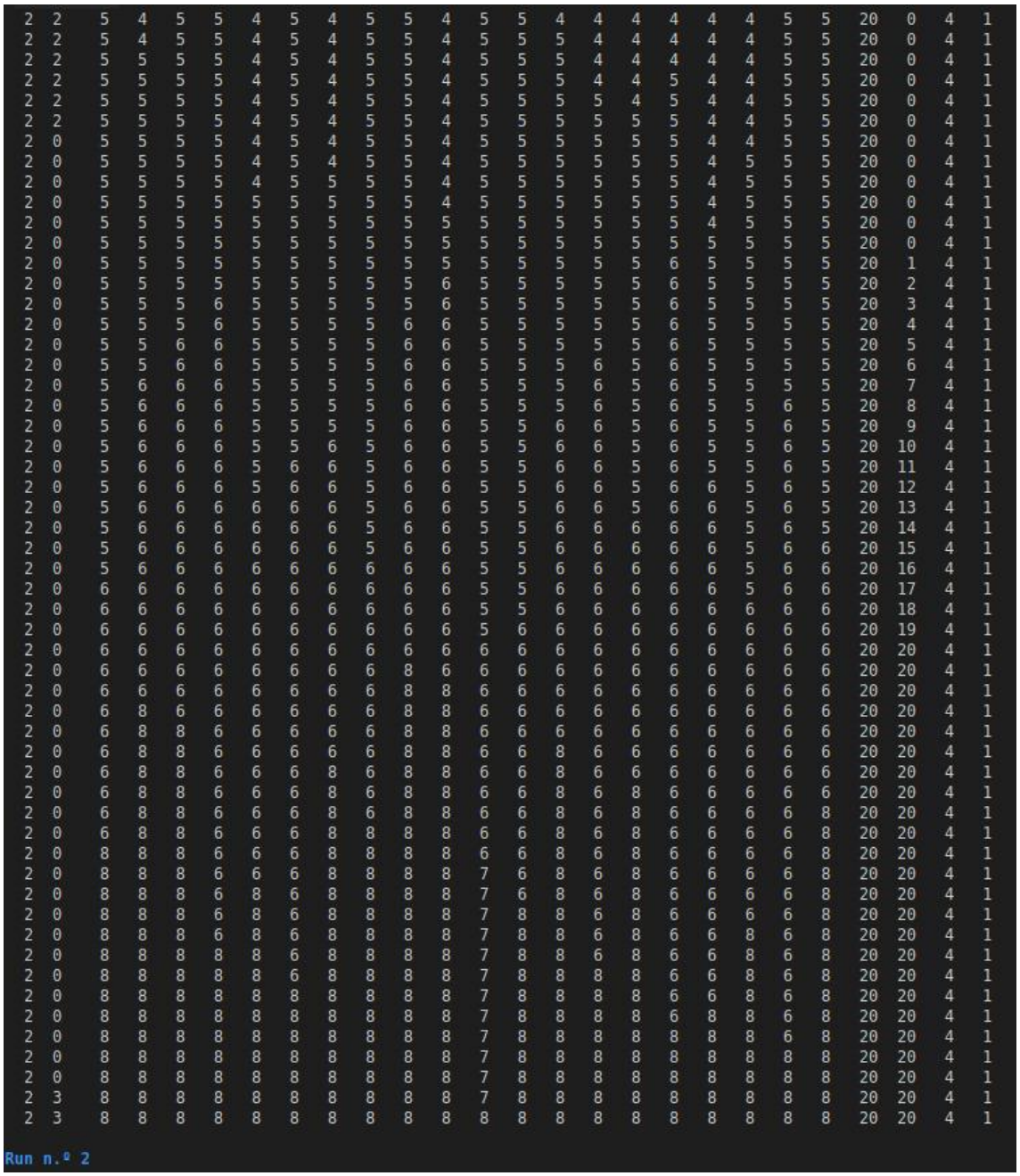
Ao correr “`run.sh`”:

```
TERMINAL   PROBLEMS   OUTPUT   DEBUG CONSOLE
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   2   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1    1   0   4  -1
  0  0    1   1   1   1   2   1   1   1   1   1   1   2   1   1   1   1   1   1   1   1    2   0   4  -1
  0  0    1   1   1   1   2   1   1   2   1   1   1   2   1   1   1   1   1   1   1   1    3   0   4  -1
  0  0    1   1   1   2   2   1   1   2   1   1   1   2   1   1   1   1   1   1   1   1    4   0   4  -1
  0  0    1   1   1   2   2   1   1   2   1   1   2   2   1   1   1   1   1   1   1   1    5   0   4  -1
  0  0    1   1   2   2   2   1   1   2   1   1   2   2   1   1   1   1   1   1   1   1    6   0   4  -1
  0  0    1   1   2   2   2   2   1   2   1   1   2   2   1   1   1   1   1   1   1   1    7   0   4  -1
  0  0    1   1   2   2   2   2   1   2   1   1   2   2   2   1   1   1   1   1   1   1    8   0   4  -1
  0  0    1   1   2   2   2   2   1   2   1   1   2   2   2   1   1   1   1   1   1   2    9   0   4  -1
  0  0    1   1   2   2   2   2   1   2   2   1   2   2   2   1   1   1   1   1   1   2   10   0   4  -1
  0  0    1   1   2   2   2   2   1   2   2   1   2   2   2   1   1   2   1   1   1   2   11   0   4  -1
  0  0    1   1   2   2   2   2   1   2   2   1   2   2   2   1   1   2   1   1   2   2   12   0   4  -1
  0  0    2   1   2   2   2   2   1   2   2   1   2   2   2   1   1   2   1   1   2   2   13   0   4  -1
  0  0    2   1   2   2   2   2   1   2   2   1   2   2   2   1   2   2   1   1   2   2   14   0   4  -1
  0  0    2   1   2   2   2   2   1   2   2   1   2   2   2   1   2   2   1   2   2   2   15   0   4  -1
  0  0    2   1   2   2   2   2   2   2   2   1   2   2   2   1   2   2   1   2   2   2   16   0   4  -1
  0  0    2   1   2   2   2   2   2   2   2   1   2   2   2   2   2   2   1   2   2   2   17   0   4  -1
  0  0    2   1   2   2   2   2   2   2   2   2   2   2   2   2   2   2   1   2   2   2   18   0   4  -1
  0  0    2   1   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   19   0   4  -1
  0  0    2   4   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   20   0   4   1
  0  0    2   4   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   4   20   0   4   1
  0  0    2   4   2   2   3   2   2   2   2   2   2   2   2   2   2   2   2   2   2   4   20   0   4   1
  0  0    2   4   4   2   3   2   2   2   2   2   2   2   2   2   2   2   2   2   2   4   20   0   4   1
  0  0    2   4   4   2   3   2   2   2   4   2   2   2   2   2   2   2   2   2   2   4   20   0   4   1
  0  0    2   4   4   2   3   2   2   2   4   2   2   2   2   2   2   4   2   2   2   4   20   0   4   1
  0  0    2   4   4   2   3   2   2   2   4   2   2   2   4   2   2   4   2   2   2   4   20   0   4   1
  0  0    4   4   4   2   3   2   2   2   4   2   2   2   4   2   2   4   2   2   2   4   20   0   4   1
  0  0    4   4   4   2   3   2   2   2   4   2   2   2   4   2   4   4   2   2   2   4   20   0   4   1
  0  0    4   4   4   2   3   2   2   2   4   2   2   2   4   2   4   4   2   2   4   4   20   0   4   1
  0  0    4   4   4   4   3   2   2   2   4   2   2   2   4   2   4   4   2   2   4   4   20   0   4   1
  0  0    4   4   4   4   3   4   2   2   4   2   2   2   4   2   4   4   2   2   4   4   20   0   4   1
  0  0    4   4   4   4   3   4   2   4   4   2   2   2   4   2   4   4   2   2   4   4   20   0   4   1
  0  0    4   4   4   4   3   4   2   4   4   2   4   2   4   2   4   4   2   2   4   4   20   0   4   1
  0  0    4   4   4   4   3   4   2   4   4   2   4   4   4   2   4   4   2   2   4   4   20   0   4   1
  0  0    4   4   4   4   3   4   2   4   4   2   4   4   4   2   4   4   2   2   4   4   20   0   4   1
  0  0    4   4   4   4   3   4   4   4   4   2   4   4   4   2   4   4   2   2   4   4   20   0   4   1
  0  0    4   4   4   4   3   4   4   4   4   2   4   4   4   4   4   4   2   2   4   4   20   0   4   1
  0  0    4   4   4   4   3   4   4   4   4   2   4   4   4   4   4   4   2   4   4   4   20   0   4   1
  0  0    4   4   4   4   3   4   4   4   4   4   4   4   4   4   4   4   2   4   4   4   20   0   4   1
  0  0    4   4   4   4   3   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0   4   1
  0  0    4   4   4   4   3   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0   4   1
  0  1    4   4   4   4   3   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0   4   1
  0  0    4   4   4   4   3   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0   4   1
  1  0    4   4   4   4   3   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0   4   1
  1  0    4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0   4   1
  2  0    4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0   4   1
  2  0    4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0   4   1
  2  2    4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0   4   1
  2  2    4   4   5   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   20   0   4   1
  2  2    4   4   5   4   4   4   4   4   5   4   4   4   4   4   4   4   4   4   4   4   20   0   4   1
  2  2    4   4   5   4   4   4   4   4   5   4   4   4   4   4   4   4   4   4   4   5   20   0   4   1
  2  2    4   4   5   5   4   4   4   4   5   4   4   4   4   4   4   4   4   4   4   5   20   0   4   1
  2  2    4   4   5   5   4   4   4   4   5   4   4   4   4   4   4   4   4   4   5   5   20   0   4   1
  2  2    4   4   5   5   4   4   4   4   5   4   5   4   4   4   4   4   4   4   5   5   20   0   4   1
  2  2    4   4   5   5   4   4   4   4   5   4   5   5   4   4   4   4   4   4   5   5   20   0   4   1
  2  2    5   4   5   5   4   4   4   4   5   4   5   5   4   4   4   4   4   4   5   5   20   0   4   1
  2  2    5   4   5   5   4   5   4   4   5   4   5   5   4   4   4   4   4   4   5   5   20   0   4   1
```

(continua na página seguinte)

(continuação)



E vai continuar a correr, parar arranjar mil soluções.

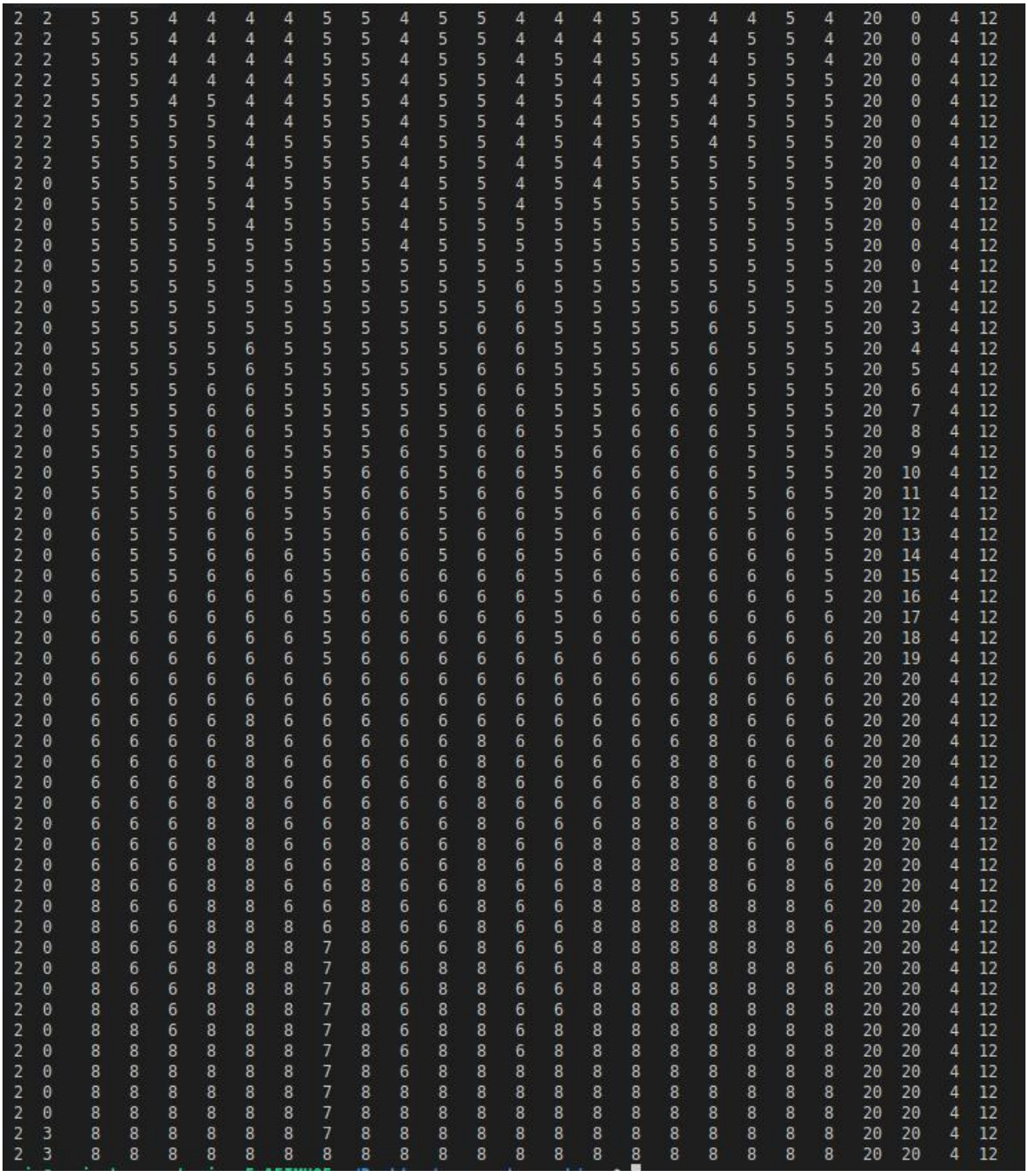
Ao correr “semSharedMemRestaurant”:

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

(continua na página seguinte)

(continuação)



Conclusão

Este relatório foi feito com base no código que o acompanha. Por sua vez, este código foi efetuado com base no código (incompleto) previamente fornecido na página da cadeira de Sistemas Operativos no website do *eLearning*. Tendo como ponto de partida esse código já fornecido efetuámos algumas alterações de modo a completá-lo, inserindo código nas partes indicadas pelos comentários. Deste modo, ao corrermos o código este passou-nos a dar os resultados imprimidos.

Com a realização deste trabalho conseguimos aprofundar os nossos conhecimentos relativamente aos mecanismos associados à execução e sincronização de "*threads*". Conseguimos ainda aprofundar os nossos conhecimentos na linguagem C, assim como no funcionamento dos semáforos e na memória partilhada, entre outros.

Para conseguirmos concluir este trabalho tivemos de, primeiramente, perceber o funcionamento do código já dado, ou seja, entender como é que as funções necessárias para este trabalho funcionavam e qual o objetivo de cada uma. Depois de percebido, completámos o código, obedecendo à estrutura e às regras necessárias para que este corresse corretamente.

Por fim, verificámos novamente o código todo e verificámos ainda se os resultados obtidos faziam sentido relativamente ao pedido inicialmente neste trabalho.

Fontes

Para realizarmos este trabalho com sucesso recorremos a algum material fornecido na página do *eLearning* desta cadeira (Sistemas Operativos).

Imagem utilizada na capa do relatório:
https://c.stocksy.com/a/5CG900/z9/2207205.jpg