Índice

Introdução

O segundo trabalho prático de Sistemas Operativos consiste na simulação de um restaurante com três entidades diferentes: o(s) *client(s)*, o *waiter* e o *chef*.

Este projeto, desenvolvido tendo como base a linguagem C, pretende demonstrar o funcionamento, manipulação e sincronização de semáforos com o intuito de obter a solução desejada.

Com o intuito de visualizar o resultado da execução dos processos pretendidos, após compilação deste, corremos o programa *./probSemSharedRestaurant*.

Variáveis gerais e de estados

No programa probConst.h temos as variáveis necessárias para o desenvolvimento do programa.

As variáveis gerais ditam o número máximo de clientes no restaurante (TABLESIZE), o máximo de tempo para estes comerem (MAXEAT), e o máximo de tempo para o chef fazer os pedidos (MAXCOOK).

Além disso, existem três variáveis associadas ao waiter que ditam, consoante o valor de retorno da função, se este deve levar o pedido ao chef (FOOD REQ), se deve levar o pedido para a mesa (FOODREADY) ou se deve processar o pagamento (BILL). Os *clients*, o *chef* e o *waiter* podem ter vários estados possíveis.

Do *waiter*: à espera do pedido (WAIT_FOR_REQUEST), o *waiter* leva o pedido ao *chef* (INFORM_CHEF), leva o pedido para os *clients* (TAKE_TO_TABLE), recebe pagamento (RECEIVE_PAYMENT).

Do *chef*: à espera de pedidos (WAIT_FOR_ORDER), confecionando os pedidos (COOK) e descansando quando acaba o serviço (REST).

Do *client*: indo para o restaurante (INIT), esperando pelo resto das pessoas (WAIT_FOR_FRIENDS), pedindo o que pretendem comer (FOOD_REQUEST), pedido efetuado apenas pela primeira pessoa a chegar ao restaurante; esperando pela comida (WAIT_FOR_FOOD), comendo (EAT), esperando pelos outros acabarem de comer (WAIT_FOR_OTHERS), esperando pelo pagamento (WAIT_FOR_BILL), pago pelo último cliente a chegar ao restaurante; e quando acaba a refeição (FINISHED).

Para uma melhor percepção dos diferentes estados destas variáveis, estes foram agrupados em tabelas.

Variáveis Gerais

Variável	**Valor**
TABLESIZE	20
MAXEAT	500000
MAXCOOK	3000000

Variáveis do Waiter

Variável	Valor
FOODREQ	1
FOODREADY	2
BILL	3

Estado do *waiter*

Estado	Valor
WAIT_FOR_REQUEST	0
INFORM_CHEF	1
TAKE_TO_TABLE	2
RECEIVE_PAYMENT	3

Estado do *chef*

Estado	Valor
WAIT_FOR_ORDER	0
COOK	1
REST	2

Estado do(s) *client(s)*

Estado	Valor
INIT	1
WAIT_FOR_FRIENDS	2
FOOD_REQUEST	3
WAIT_FOR_FOOD	4
EAT	5
WAIT_FOR_OTHERS	6
WAIT_FOR_BILL	7
FINISHED	8

Estrutura de Dados Internos

No ficheiro probDataStruct, estão contidas as estruturas de dados essenciais para o programa. Uma destas será o tipo STAT, uma estrutura que vai armazenar os estados de todas as entidades. Deste modo, este possui três parâmetros: um unsigned inteiro para guardar o estado do empregado, um unsigned inteiro para guardar o estado da chefe e um array de inteiros unsigned clientStat para guardar o estado dos clientes, array este com tamanho igual ao máximo de clientes possível.

```
typedef struct {
    /** \brief waiter state */
    unsigned int waiterStat;
    /** \brief chef state */
    unsigned int chefStat;
    /** \brief client state array */
    unsigned int clientStat[TABLESIZE];

} STAT;
```

O tipo de dados FULL_STAT vai guardar informação sobre o resto das entidades que participam no problema. Este guarda a estrutura de dados STAT apresentada anteriormente numa variável st, tendo informação sobre todos os estados atuais, guarda o número de clientes na mesa, o número de clientes que acabaram de comer, o id do primeiro cliente a chegar ao restaurante e o id do último cliente a chegar ao restaurante. Além disto, este vai utilizar 4 flags: *foodRequest*, do cliente para indicar ao empregado quando o cliente quer fazer o pedido, *foodOrder*, do empregado para indicar ao chefe quando começar a cozinhar, *foodReady*, do chefe para indicar ao empregado que a comida está pronta (e que pode levar para a mesa) e *paymentRequest*, do cliente para indicar ao empregado quando quer pagar a refeição.

```
typedef struct
{   /** \brief state of all intervening entities */
    STAT st;

    /** \brief number of clients at table */
    int tableClients;
    /** \brief number of clients that finished eating */
    int tableFinishEat;

    /** \brief flag of food request from client to waiter */
    int foodRequest;
    /** \brief flag of food order from waiter to chef */
    int foodOrder;
    /** \brief flag of food ready from chef to waiter */
    int foodReady;
    /** \brief flag of payment request from client to waiter */
    int paymentRequest;

    /** \brief id of first client to arrive */
    int tableLast;
    /** \brief id of last client to arrive */
    int tableFirst;

} FULL_STAT;
```

Definição de Semáforos

De modo que um processo ou condição seja executada antes de outra, é necessário usar semáforos. Estes permitem a correta execução dos eventos, bloqueando processos quando necessário.

A estrutura de dados SHARED_DATA contém uma variável fSt do tipo FULL_STAT e uma série de semáforos:

Semáforo	Propósito
mutex (valor inicial=1)	Quando entra na região crítica este é decrementado através do semDown(), bloqueando outros processos que tentem entrar na região. À saída desta região esta é incrementada através do semUp().
friendsArrived (valor inicial =0)	Utilizado para bloquear o processo dos clientes enquanto estes esperam pela mesa ficar cheia, altura em que o semáforo é incrementado e o processo é libertado.
requestReceived (valor inicial=0)	Utilizado para bloquear o processo dos clientes enquanto espera a verificação feita pelo empregado para processar o pedido, altura em que o semáforo é incrementado e o processo é libertado.
foodArrived (valor inicial=0)	Utilizado para bloquear o processo dos clientes enquanto esperam pela comida, altura em que o semáforo é incrementado e o processo é libertado.
allFinished (valor inicial=0)	Utilizado para bloquear o processo dos clientes até todos os clientes acabarem de comer, altura em que o semáforo é incrementado e o processo é libertado.
waiterRequest (valor inicial=0)	Utilizado para bloquear o processo do waiter até que o pedido esteja feito, altura em que o semáforo é incrementado e o processo é libertado.
waitorder (valor inicial=0)	Utilizado para bloquear o processo do chef até que o pedido lhe seja feito pelo waiter, altura em que o semáforo é incrementado e o processo é libertado.

Personagens

Client (SemSharedClient.c)

1. **travel**

   Esta função, definida inicialmente, define o tempo que os clientes levam para chegar ao restaurante, com tempo máximo de 1000000.

```c
static void travel (int id)
{
    usleep((unsigned int) floor ((1000000 * random ()) / RAND_MAX + 1000));
}
```

2. **eat**

   Esta função define o tempo que os clientes levam para comer a refeição, este vai ter um tempo máximo MAXEAT.

```c
static void eat (int id)
{
    usleep((unsigned int) floor ((MAXEAT * random ()) / RAND_MAX + 1000));
}
```

3. **waitFriends**

   Nesta função, é definido o estado inicial de cada cliente (INIT), significando que ainda não chegou ao Restaurante. Consoante estes vão entrando, o contador *tableClients* irá ser incrementado, e se este valor for igual a 1 (primeiro cliente), *tableFirst* será igual ao *id* deste.

   Em seguida, se o número de pessoas na mesa (*tableClients*) for diferente do tamanho máximo da mesa (TABLESIZE), o estado do cliente que passa será mudado para “a esperar pelos amigos” (WAIT_FOR_FRIENDS). Caso esta condição não se verifique (último elemento a chegar), *tableLast* será igual ao *id* deste último cliente e o estado deste é mudado para “a esperar pelos amigos” (WAIT_FOR_FRIENDS) e o semáforo *friendsArrived* será decrementado, num ciclo *for* (para todos os clientes), significando que todos os amigos chegaram.

   Dentro do *else* ainda, a variável booleana *first* é mudada para *true*, de modo a fazer o pedido, já que todos os clientes chegaram. Antes deste executar a condição *else*, irá ser incrementado o semáforo *friendsArrived*, para que estes fiquem à espera dos outros clientes de modo que a mesa fique cheia.

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

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

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

    sh->fSt.tableClients++;

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

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

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

        first = true;
    }

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

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

    return first;
}
```

4. **orderFood**

Nesta função será feito o pedido, somente, pela primeira pessoa a entrar no restaurante (*tableFirst*). Deste modo, a flag *foodRequest* irá ser incrementada para 1 (*foodRequest = 1*), e o semáforo *waiterRequest* será decrementado, significando que o waiter pode deixar de estar à espera.

Seguidamente, o estado do primeiro cliente (*tableFirst*) será mudado para FOOD_REQUEST, e o semáforo *requestReceived* irá ser decrementado, ao executar *semDown*, bloquear o processo dos clientes enquanto espera a verificação feita pelo empregado.

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

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

5. **waitFood**

Nesta função o cliente irá esperar pela comida e quando a comida chegar irá comer a refeição pedida. Assim, se o *id* do cliente for diferente do que entra primeiro no restaurante (*tableFirst*), e caso a condição seja verdadeira, o estado será mudado para WAIT_FOR_FOOD.

Depois, será decrementado o semáforo *foodArrived* bloqueando o processo dos clientes enquanto esperam pela comida.

Por fim, o estado de todos os clientes será mudado para EAT.

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

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

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

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

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

6. waitAndPay

Nesta função os clientes, consoante vão acabando de comer, ficam à espera que os outros acabem de comer para que o último a entrar no restaurante (*tableLast*) faça o pagamento. No fim do pagamento, os clientes podem sair do restaurante.

Primeiramente, irá ser incrementado o *tableFinishEat* indicando quantas pessoas acabaram de comer (depois de passarem pela função eat, e esperarem o tempo que demoram a comer). O estado dos clientes é mudado para WAIT_FOR_OTHERS.

Seguidamente, será verificado se o número de clientes que acabaram de comer (*tableFinishEat*) é igual ao tamanho da mesa (TABLESIZE), e caso esta condição se verifique, significando que toda a gente acabou de comer, o semáforo *allFinished* irá ser incrementado, libertando o processo dos *clients* e alterando o valor da variável booleana *last* para *true*.

Posteriormente, esta função irá verificar o valor de *last*, que se for *true*, irá mudar a flag *paymentRequest* para 1 (*paymentRequest = 1*) e mudar o estado do último cliente a chegar ao restaurante (*tableLast*) para WAIT_FOR_BILL. O semáforo *waiterRequest* será incrementado, libertando o processo do waiter para ele receber o pedido do pagamento.

De seguida, o semáforo *allFinished* será incrementado por pessoa, enquanto toda a gente na mesa não tiver acabado de comer, libertando estes processos para a próxima iteração.

Por fim, se o *id* do cliente for diferente do último cliente que chega ao restaurante (*tableLast*), pois é quem paga a refeição, e se o número de pessoas que acabaram de comer (*tableFinishEat*) for igual ao tamanho da mesa (TABLESIZE), o estado do cliente será mudado para FINISHED.

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

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

    /* insert your code here */
    if(sh->fSt.tableFinishEat == TABLESIZE){
        if (semUp (semgid, sh->allFinished) == -1) {                                    /* enter critical region */
            perror ("error on the down operation for semaphore access (CT)");
            exit (EXIT_FAILURE);
        }
        last = true;
    }

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

    if(last) {
        if (semDown (semgid, sh->mutex) == -1) {                                    /* enter critical region */
           perror ("error on the down operation for semaphore access (CT)");
           exit (EXIT_FAILURE);
        }
        /* insert your code here */
        sh->fSt.paymentRequest=1;
        sh->fSt.st.clientStat[sh->fSt.tableLast] = WAIT_FOR_BILL;
        saveState(nFic, &sh->fSt);
        if (semUp (semgid, sh->waiterRequest) == -1) {                                      /* enter critical region */
            perror ("error on the down operation for semaphore access (CT)");
            exit (EXIT_FAILURE);
        }
    }

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

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

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

    /* insert your code here */
    if(id!=sh->fSt.tableLast && sh->fSt.tableFinishEat==TABLESIZE){
        sh->fSt.st.clientStat[id]=FINISHED;
        saveState(nFic, &sh->fSt);
    }

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

Waiter (SemSharedWaiter.c)

1. waitForClientOrChef

Esta função irá retratar a espera do waiter pelo cliente, quer seja ao efetuar o pedido quer seja a efetuar o pagamento, e a espera do waiter pelo chef, até que a comida esteja pronta.

Assim, irá inicializar-se o waiter com o estado WAIT_FOR_REQUEST e o semáforo *waiterRequest* será decrementado, bloqueando o processo do waiter até que o pedido esteja feito.

Posteriormente, serão verificadas 3 condições tendo em conta as *flags*, consoante o tipo de pedido feito ao waiter. Em qualquer uma das condições a verificar, o semáforo *requestReceived* será incrementado em dois dos três *ifs*, quando o pedido é feito (*foodRequest = 1*), e quando este é levado à mesa (*paymentRequest = 1*) significando que os processos dos clientes serão libertados. O mesmo não irá acontecer em *foodReady* pois este semáforo retrata, unicamente, a espera dos clientes pelo waiter, o que só vai acontecer quando este recebe os pedidos, quer seja pedido da comida, quer seja de pagamento.

Caso a *flag foodRequest* esteja a 1 (*foodRequest = 1*), que acontece quando o cliente faz um pedido, esta será retornada para 0 pois o pedido já está feito. Assim, o estado do cliente que fez o pedido, o primeiro a entrar no restaurante (*tableFirst*), será mudado para WAIT_FOOD e a função retornará FOODREQ.

Se a flag *foodReady* for igual a 1 (*foodReady = 1*), significando que a comida está pronta, a flag *foodReady* retornará a 0 e a função retornará FOODREADY.

Porém, se a flag *paymentRequest* tiver a 1 (*paymentRequest = 1*), retornará a 0, uma vez que o pedido de pagamento já foi feito, retornando a função, por sua vez, BILL.

A variável *ret* é declarada como *int*, porém os valores de retorno aparentam ser *strings*, isto pois estes valores de retorno FOODREQ, FOODREADY e BILL estão definidos inicialmente por três inteiros 1, 2 e 3 respetivamente.

```
static int waitForClientOrChef()
{
    int ret=0;
    if (semDown (semgid, sh->mutex) == -1)  {                                          /* enter critical region */
        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 up operation for semaphore access (WT)");
        exit (EXIT_FAILURE);
    }
    /* insert your code here */
    if (semDown (semgid, sh->waiterRequest) == -1) {                                   /* enter critical region */
        perror ("error on the down operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
    }
```

```
    if (semDown (semgid, sh->mutex) == -1)  {                                   /* enter critical region */
        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) {                                    /* enter critical region */
            perror ("error on the down operation for semaphore access (CT)");
            exit (EXIT_FAILURE);
        }
        sh->fSt.foodRequest = 0;
        sh->fSt.st.clientStat[sh->fSt.tableFirst] = WAIT_FOR_FOOD;
        saveState(nFic, &sh->fSt);
        ret=FOODREQ;
    } else if(sh->fSt.foodReady==1){
        sh->fSt.foodReady=0;
        ret=FOODREADY;
    } else if(sh->fSt.paymentRequest==1){
        if (semUp (semgid, sh->requestReceived) == -1) {                                    /* enter critical region */
            perror ("error on the down operation for semaphore access (CT)");
            exit (EXIT_FAILURE);
        }
        ret=BILL;
    }

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

    return ret;
}
```

2. informChef

Esta função servirá para informar o chef de que pode começar a cozinhar.

A primeira mudança nesta função é a alteração do estado do empregado que passa a informar o chefe de quais são os pedidos dos clientes (INFORM_CHEF).

Incrementa-se o semáforo *waitOrder* dado que o empregado já está no processo de informar o chefe, assim, o cozinheiro já não precisa de esperar mais pelos pedidos.

Por consequência, ativa-se a flag *foodOrder* (*foodOrder = 1*), ou seja, o pedido passa do empregado para o chefe.

```
static void informChef ()
{
    if (semDown (semgid, sh->mutex) == -1)  {                                   /* enter critical region */
        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);
    if (semUp (semgid, sh->waitOrder) == -1) {                                  /* enter critical region */
        perror ("error on the down operation for semaphore access (CT)");
        exit (EXIT_FAILURE);
    }

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

    /* insert your code here */
    sh->fSt.foodOrder = 1;
}
```

3. takeFoodToTable

Chegamos ao momento em que o empregado leva a comida aos clientes depois de já estar pronta.

Num ciclo *for*, até um tamanho máximo do número de pessoas presentes na mesa (todos os clientes), incrementa-se o semáforo *foodArrived* porque o tempo de espera pela comida por parte dos clientes já acabou.

Com isto, altera-se o estado do empregado para levar a comida à mesa (TAKE_TO_TABLE) e depois, salva-se o estado.

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

    /* insert your code here */
    for(int i=0; i<TABLESIZE;i++) {
        if (semUp (semgid, sh->foodArrived) == -1) {                                    /* enter critical region */
            perror ("error on the down operation for semaphore access (CT)");
            exit (EXIT_FAILURE);
        }
    }

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


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

4. receivePayment

Esta função retrata o empregado a receber o pagamento e o último cliente a sair do restaurante.

Depois do último cliente a chegar ao restaurante (*tableLast*), requisitar o pagamento ao empregado, é chamada a função *receivePayment*. Uma vez que o cliente já requisitou o pagamento, desativamos a flag *paymentRequest* (*paymentRequest = 0*). Alteramos, ainda, o estado do empregado de mesa passando a receber o pagamento (RECEIVE_PAYMENT).

Para finalizar o jantar de amigos, e enquanto os clientes que não pagaram já estarem no estado de finalização, colocamos o cliente que pagou no estado final (*sh->fSt.st.clientStat[sh->fSt.tableLast] = FINISHED*).

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

    /* insert your code here */
    sh->fSt.paymentRequest = 0;
    sh->fSt.st.waiterStat = RECEIVE_PAYMENT;
    saveState(nFic, &sh->fSt);
    sh->fSt.st.clientStat[sh->fSt.tableLast]=FINISHED;
    saveState(nFic, &sh->fSt);

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

Chef (SemSharedChef.c)

1. waitForOrder

Esta função retrata a espera do chef por um pedido por parte do waiter.

O estado do chef é inicializado fora de função onde se define que este espera pelo pedido de comida que lhe será entregue pelo empregado (WAIT_FOR_ORDER).

Na função *waitForOrder*, começamos por decrementar o semáforo *waitOrder* de forma a indicar que o cozinheiro está à espera do pedido.

Seguidamente, desligamos a flag *foodOrder* (*foodOrder = 0*) assim que o chefe recebe o pedido do empregado.

Por fim, uma vez que o chefe já recebeu o pedido da comida do empregado de mesa, o estado do cozinheiro muda para cozinhar (COOK).

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

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

2. processOrder

É a vez do cozinheiro processar o pedido (fazer a comida) e depois de a entregar ao empregado ir descansar.

O cozinheiro, ainda no estado de cozinhar (COOK), entra na *processOrder* e é gerado um tempo de cozedura aleatório para cada refeição hipotética.

Depois das refeições estarem prontas, a flag *foodReady* é ativada (*foodReady = 1*) o que indica que a comida está pronta e pode ser entregue ao empregado de mesa para a levar até à mesa. Consequentemente, o estado do chefe passa a descanso (REST).

Finalmente, incrementa-se o semáforo *waiterRequest* o que significa que o empregado já não está mais à espera dos pedidos porque já estão prontos.

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

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

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

Demonstrações

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

JANTAR DE AMIGOS (RESTAURANT)

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

Demonstração 2

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

JANTAR DE AMIGOS (RESTAURANT)

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

Demonstração 3

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

JANTAR DE AMIGOS (RESTAURANT)

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

Conclusão

A resolução deste trabalho foi deveras educativa ao aprofundar os nossos conhecimentos sobre semáforos e a sua utilização, quer a partir do material teórico e prático das aulas de Sistemas Operativos, quer a partir da pesquisa impulsionada pelos obstáculos que surgiram na realização deste trabalho.

Este trabalho foi um sucesso, pois os resultados obtidos são idênticos aos esperados, tendo a realização deste nos motivado para expandirmos os nossos horizontes sobre semáforos.

Bibliografia

-Slides Teóricos de Sistemas Operativos
-https://openai.com/dall-e-2/ (Imagem Capa)