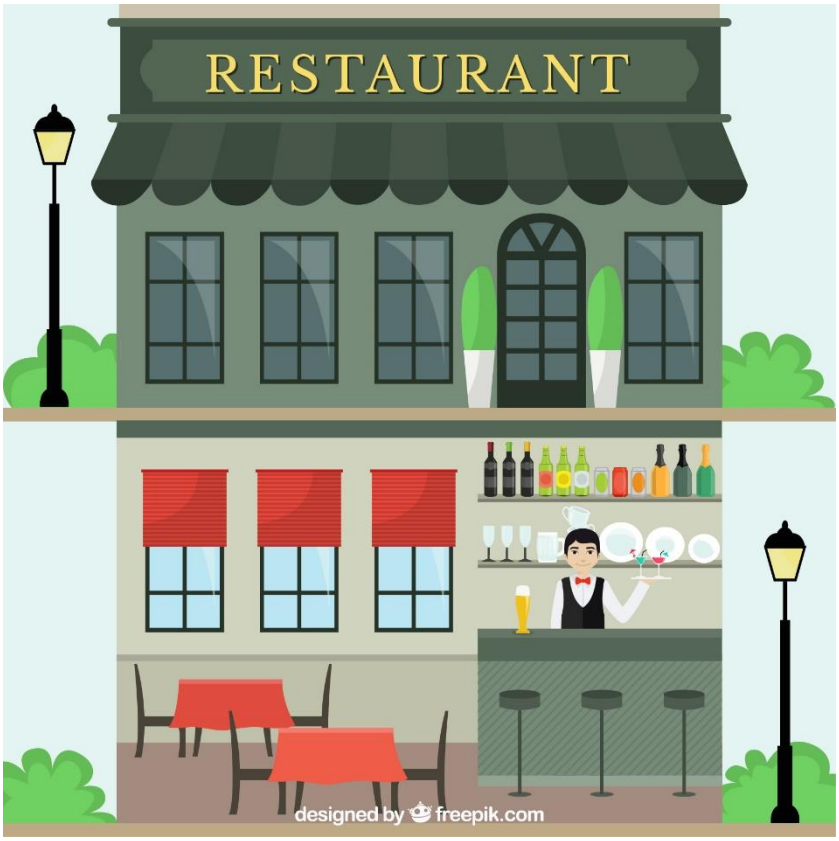
universidade de aveiro
deti departamento de eletrónica, telecomunicações e informática

Atribuição 2: Compreensão dos mecanismos associados à execução e sincronização de processos e *threads*.

RESTAURANTE



Sistemas Operativos

Trabalho realizado por:

João Miguel Ferreira Varela, n.ºMec: 113780

Carolina Marques Prata, n.ºMec: 114246

Ano letivo: 2023/2024

Índice

1.Introdução

No âmbito da disciplina de Sistemas Operativos, foi-nos proposto como segundo trabalho prático, uma simulação de um restaurante com quatro entidades distintas, *Groups, Waiter, Receptionist* e *Chef.* Cada uma dessas entidades são processos independentes e a sua sincronização será realizada através de semáforos e da memória partilhada.

Durante a execução deste trabalho tivemos especial atenção ao acesso à zona crítica, impedindo que dois processos acedessem, simultaneamente, à memória partilhada, o que implicaria desconcordâncias no programa.

A apresentação do funcionamento deste restaurante é realizada mediante a impressão de uma tabela devidamente formatada que atualiza os estados dos intervenientes (*Groups, Waiter, Receptionist* e *Chef*) no decorrer do jantar.

Este relatório tem como objetivo explicitar todo o raciocínio que nos levou a formular o código necessário à boa execução e sincronização de todos os processos intervenientes.

2. Estados dos Intervenientes

Estado	Valor	Significado
	chefStat	
WAIT_FOR_ORDER	0	Inicialmente o *chef* está pronto a receber um pedido de comida para confecionar, que lhe será pedido pelo waiter
COOK	1	Após receber o pedido, o chef começa a cozinhar o pedido
	groupStat[id]	
GOTOREST	1	Estado inicial, onde se encontra o group
ATRECEPTION	2	O group espera na receção e realiza um pedido de mesa ao rececionista. Se as duas mesas estiverem ocupadas, o group permanecerá na receção à espera, caso contrário ser-lhe-á atribuída uma mesa
FOOD_REQUEST	3	Após estarem com mesa atribuída, o group realiza o seu pedido de comida ao waiter**.**
WAIT_FOR_FOOD	4	O *group* espera na mesa, até o seu pedido estar pronto.
EAT	5	Após a entrega da refeição, pelo *waiter*, o group começa a comer.
CHECKOUT	6	O *group* espera pela disponibilidade do *receptionist* para realizar o Checkout. Quando este se encontrar disponível, solicita-lhe o pedido de pagamento.
LEAVING	7	Ao fim do pagamento, o group sai do restaurante.
	waiterStat	
WAIT_FOR_REQUEST	0	O *waiter* espera pelo pedido de comida do *group*
INFORM_CHEF	1	Após receber o pedido, dirige-se ao c*hef* para lhe indicar a refeição
TAKE_TO_TABLE	1	Quando a comida estiver pronta, o *waiter* leva o pedido à mesa
	receptionistStat	
WAIT_FOR_REQUEST	0	O *receptionist*, caso esteja disponível, aguarda pelo pedido do *group*, podendo este ser um pedido de uma mesa ou um pedido de pagamento.
ASSIGNTABLE	1	Caso o pedido requisitado pelo *group* seja de uma mesa, o *receptionist* atribui-lhes uma mesa.
RECVPAY	2	Na eventualidade de o *group* necessitar de pagar, o *receptionist* recebe o devido pagamento.

3. Significado de cada semáforo a utilizar

Semáforo	Objetivo
Mutex (valor inicial = 1)	Identifica uma determinada região crítica, que bloqueia qualquer processo que tente entrar numa região de modo a proteger a execução do processo atual. Dentro desta também é atualizado os estados dos diversos intervenientes.
ReceptionistReq (valor inicial = 0)	Utilizado pelo *receptionist* para esperar pelos *groups*.
receptionistRequestPossible (valor inicial = 1)	Tem o intuito de informar os *groups* quando é que o *receptionist* está disponível para receber um pedido do *group*.
waiterRequest (valor inicial = 0)	Utilizado pelo *waiter* a fim de ter a informação de quando é chamado tanto pelos groups como pelo chef.
waiterRequestPossible (valor inicial = 1)	Usado pelos *groups* e *chef* antes de realizarem um pedido ao *waiter*, verificando a disponibilidade deste.
waitOrder (valor inicial = 0)	Informa o *chef* da chegada de um pedido através do *waiter*.
orderReceived (valor inicial = 0)	Utilizado pelo *waiter* para ser informado da receção do pedido pelo *chef*.
waitForTable[MAXGROUPS] (valor inicial = 0)	Informa o *group* caso este tenha de aguardar por uma mesa
requestReceived[NUMTABLES] (valor inicial = 0)	Utilizado pelo *group* para esperar pelo *waiter* de modo a realizar o pedido.
foodArrived[NUMTABLES] (valor inicial = 0)	Tem o intuito de informar os *groups* que o seu pedido chegou à mesa.
tableDone (valor inicial = 0)	Usado pelos *groups* para aguardar a possibilidade de realizar o pagamento.

4. Estrutura do Código

Nesta parte do relatório explicaremos o raciocínio que nos fez chegar à implementação final. Explanaremos, detalhadamente, cada parte de código que tínhamos de preencher.

4.1 SemSharedMemChef.c

4.1.1 Função waitForOrder()

A função *waitForOrder* é destinada ao *chef* para aguardar e processar um pedido de comida.

- **Propósito da Função:**

Esperar por um Pedido de Comida: O *chef* aguarda um sinal do *waiter* que indica que há um pedido de comida para ser preparado.

Atualizar e salvar o Estado Interno: Quando o *chef* recebe um pedido, ele atualiza seu estado (COOK), pois começou a preparar o pedido.

Reconhecer o Pedido Recebido: Após receber e processar o pedido, o *chef* deve sinalizar que o pedido foi recebido e está sendo atendido.

- **Funcionamento da Função:**

Esperar pelo Sinal de Pedido de Comida: semDown(semgid, sh->waitOrder): Espera pelo sinal (waitOrder) do *waiter,* que indica que um pedido foi feito. Isto é uma operação de "down" em um semáforo, o que faz o *chef* aguardar até que o *waiter* sinalize que há um pedido.

```
//TODO insert your code here
//The chef waits for the food request that will be provided by the waiter.
if (semDown (semgid, sh->waitOrder) == -1) {
    perror ("error on the up operation for semaphore access (PT)");
    exit (EXIT_FAILURE);
}
```

Processar o Pedido: Dentro da região crítica, o *chef* lê o *group* que fez o pedido (lastGroup=sh->fSt.foodGroup) e atualiza o estado para COOK. saveState(nFic, &sh->fSt): Salva o estado atual do *chef*, o que é importante para registar a progressão do pedido.

```
//TODO insert your code here
//Updates its state and saves internal state.
lastGroup=sh->fSt.foodGroup;
sh->fSt.st.chefStat = COOK;
saveState(nFic, &sh->fSt);
```

Reconhecer o Pedido: semUp(semgid, sh->orderReceived): Sinaliza que o pedido foi recebido e está a ser processado. Notifica que o *chef* começou a preparar o pedido.

```
//TODO insert your code here
// Received order should be acknowledged
if (semUp (semgid, sh->orderReceived) == -1) {
    perror ("error on the up operation for semaphore access (PT)");
    exit (EXIT_FAILURE);
}
```

4.1.2 Função processOrder()

A função *processOrder* destina-se ao *chef* para o processo de preparação de um pedido de comida e comunicação com o *waiter* assim que a comida estiver pronta.

- **Propósito da Função:**

Cozinhar o Pedido: O *chef* leva algum tempo para cozinhar o pedido recebido.

Comunicar com o *waiter*: Após terminar de cozinhar, o *chef* informa o *waiter* que a comida está pronta para ser recolhida e posteriormente, servida.

Atualizar e Salvar o Estado Interno: Durante todo o processo, o *chef* atualiza o seu estado (por exemplo, de esperar por um pedido para cozinhar, e depois para esperar pelo próximo pedido) e salva essas informações no estado interno.

- **Funcionamento da Função:**

Simular Tempo de Cozimento: usleep(...): Simula o tempo que o pedido demora a ser cozinhado.

Esperar pela Disponibilidade do *waiter*: semDown(semgid, sh->waiterRequestPossible): Antes de notificar que a comida está pronta, o *chef* verifica se o *waiter* está disponível para receber novos pedidos.

```
//TODO insert your code here

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

Atualizar Estado e Preparar Notificação: Atualiza seu estado para WAIT_FOR_ORDER e salva esse estado. Esta etapa indica que o *chef* terminou de cozinhar e está pronto para começar o próximo pedido. Notifica o *waiter* (sh->fSt.waiterRequest.reqType = FOODREADY; sh->fSt.waiterRequest.reqGroup = lastGroup;), informando que o pedido do *group* específico está pronto.

```
//TODO insert your code here

sh->fSt.st.chefStat = WAIT_FOR_ORDER;
saveState(nFic,&(sh->fSt));
sh->fSt.waiterRequest.reqType = FOODREADY;
sh->fSt.waiterRequest.reqGroup = lastGroup;
```

Informar o *waiter*: Após efetuar a notificação e sair da região crítica (semUp(semgid, sh->mutex)), o *chef* sinaliza o *waiter* (semUp(semgid, sh->waiterRequest)) que a comida está pronta para ser entregue.

```
//TODO insert your code here
//Signals the waiter that food is ready
if (semUp (semgid, sh->waiterRequest) == -1) {
    perror ("error on the down operation for semaphore access (CT)");
    exit (EXIT_FAILURE);
}
```

4.2 SemSharedMemWaiter.c

4.2.1 Função waitForClientOrChef()

A função waitForClientOrChef é projetada para o *waiter* aguardar e processar pedidos tanto dos *groups* quanto do *chef*.

- **Propósito da Função:**

Aguardar por Pedidos: Esperar por um pedido de um *group* ou um sinal do *chef*.

Processamento do Pedido: Ler o pedido recebido e preparar-se para a próxima ação.

- **Funcionamento da Função:**

Atualização do Estado do *waiter*: O estado do *waiter* é atualizado para WAIT_FOR_REQUEST.

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

Aguardar Pedido: semDown(semgid, sh->waiterRequest): O *waiter* aguarda por um pedido, seja de um *group* ou do *chef*.

```
// TODO insert your code here
// Waiter waits for request from group or from chef
if (semDown(semgid, sh->waiterRequest) == -1){
    perror ("error on the up operation for semaphore access (WT)");
    exit (EXIT_FAILURE);
}
```

Leitura do Pedido: req = sh->fSt.waiterRequest: O pedido é lido pelo *Waiter*.

```
// TODO insert your code here
// Waiter reads request
req = sh->fSt.waiterRequest;
saveState(nFic, &sh->fSt);
```

Sinalização de disponibilidade: semUp(semgid, sh->waiterRequestPossible): Sinaliza que está pronto para receber outro pedido.

```
// TODO insert your code here
// The waiter should signal that new requests are possible.
if (semUp (semgid, sh->waiterRequestPossible) == -1) {
    perror ("error on the down operation for semaphore access (WT)");
    exit (EXIT_FAILURE);
}
```

Retorno da Função: A função retorna o pedido (req), que contém as informações do pedido feito pelo group ou pelo chef.

4.2.2 Função informChef()

A função informChef informa o *chef* sobre um pedido de comida.

- **Propósito da Função:**

Informar o Chef sobre o Pedido de Comida: O *waiter* atualiza o seu estado para INFORM_CHEF.

Notificar o Group: O *waiter* deve informar o *group* que seu pedido foi recebido e está a ser processado.

Aguardar Confirmação do *Chef*: O *waiter* aguarda uma confirmação de que o *chef* recebeu o pedido.

- **Funcionamento da Função:**

Atualização do Estado do *Waiter*: sh->fSt.foodOrder = 1; sh->fSt.foodGroup = n; O pedido de comida é registado, e o *group* associado a esse pedido é identificado por um id. O estado do *waiter* é atualizado para INFORM_CHEF.

Notificar o Group: semUp(semgid,sh>requestReceived[sh>fSt.assignedTable[n]]): O *waiter* informa o *group* que o seu pedido foi recebido.

```
// TODO insert your code here
// Waiter updates state
sh->fSt.foodOrder = 1;
sh->fSt.foodGroup=n;
sh->fSt.st.waiterStat = INFORM_CHEF;
saveState(nFic, &sh->fSt);

// Waiter should inform group that request is received.
if (semUp (semgid, sh->requestReceived[sh->fSt.assignedTable[n]]) == -1)   {
    perror ("error on the down operation for semaphore access (WT)");
    exit (EXIT_FAILURE);
}
```

Sinalizar ao Chef e Esperar pela Confirmação: semUp(semgid, sh->waitOrder): Sinaliza o *chef* que um novo pedido de comida foi realizado. semDown(semgid, sh->orderReceived): O *waiter* aguarda a confirmação do *chef* de que este recebeu o pedido.

```
// TODO insert your code here
// Waiter takes food request to chef
if (semUp(semgid, sh->waitOrder) == -1) {
    perror ("error on the down operation for semaphore access (WT)");
    exit (EXIT_FAILURE);
}
// Waiter should wait for chef receiving request.
if (semDown (semgid, sh->orderReceived) == -1){
    perror ("error on the up operation for semaphore access (WT)");
    exit (EXIT_FAILURE);
}
```

4.2.3 Função takeFoodToTable()

A função takeFoodToTable() é responsável por simular a ação de levar a comida até à mesa do *group*.

- **Propósito da Função:**

Levar a Comida até a Mesa do Group: Atualizar o estado do *waiter* para TAKE_TO_TABLE.

Notificar o Group sobre a Chegada da Comida: Informar o *group* que a comida chegou à mesa.

Atualizar o Estado Interno: Guardar o novo estado do *waiter* após a entrega da comida.

- **Funcionamento da Função:**

Atualização do Estado do *Waiter*: O estado do *waiter* é atualizado para TAKE_TO_TABLE, indicando que ele está a levar a comida para a mesa.

Notificar o Group da Chegada da Comida: semUp(semgid, sh->foodArrived[sh->fSt.assignedTable[n]]): Avisa o *group* na mesa sh->fSt.assignedTable[n] de que a comida chegou.

```
// TODO insert your code here
// Waiter updates state
sh->fSt.st.waiterStat = TAKE_TO_TABLE;
saveState(nFic, &sh->fSt);

// Group must be informed that food is available.
if (semUp (semgid, sh->foodArrived[sh->fSt.assignedTable[n]]) == -1)
    perror ("error on the down 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);
}
```

4.3 semSharedMemReceptionist.c

4.3.1 Função decideTableOrWait ()

Esta função indica se um *group* é imediatamente alocado a uma mesa ou se deve esperar, com base na disponibilidade das mesas.

- **Propósito da Função:**

Atribuição de mesa: O *receptionist* verifica as duas mesas, e caso uma delas se encontre livre, então é atribuída ao *group* essa mesa. A função retorna o id da mesa atribuída ao *group*, caso isto aconteça.

Decisão de Espera: Se for necessário o *group* esperar na receção por uma mesa, então será retornado –1.

- **Funcionamento da Função:**

Verificação de Validade do ID do Group: assert(n >= 0 && n < MAXGROUPS): Garante que o ID do *group* (n) é válido, isto é, está dentro do intervalo permitido de IDs de *groups*.

Alocação de Mesa: Se uma mesa livre for encontrada, a função atribui essa mesa ao *group* (sh->fSt.assignedTable[n] = i) e retorna o ID dessa mesa.

```
//TODO insert your code here
assert(n >= 0 && n < MAXGROUPS); // Id group must be valid

int tableId = -1;

for (int i = 0; i < NUMTABLES; i++) {
    if (sh->fSt.assignedTable[n] == -1) { // Check current state of tables
        tableId = i; // Select table
        sh->fSt.assignedTable[n] = i; // Update table state
        return tableId;
    }
}

// Return table id or -1 if there is no table available
return tableId;
```

4.3.2 Função decideNextGroup ()

- **Propósito da Função:**

A função decideNextGroup seleciona qual dos *groups* em espera deve ocupar uma mesa que ficou disponível.

- **Funcionamento da Função:**

Verificar Groups em Espera: É utilizado um ciclo *for* que percorre todos os groups presentes no restaurante. O primeiro a ser encontrado à espera (estado = WAIT), então irá ser o próximo *group* a ser atendido pelo *receptionist*.

Retorno da Função: No caso de não haver *groups* à espera a função devolve –1, ou no caso oposto, retorna o id do *group* que irá ser atendido de seguida.

```
//TODO insert your code here
int groupId = -1;


for (int i = 0; i < MAXGROUPS; i++) {
    if (groupRecord[i] == WAIT) { // Checks if there is a group waiting
        groupId = i; // Select group
        return groupId; // Return group id
    }
}
// Return group id or -1 if there is no group waiting
return groupId;
```

4.3.3 Função waitForGroup()

A função waitForGroup é destinada ao *receptionist* para gerenciar pedidos de *groups* que chegam ao restaurante.

– **Propósito da Função:**

O receptionist espera por um pedido de um group que chega ao restaurante. Uma vez que o pedido é recebido, o receptionist deve lê-lo e então sinalizar que está pronto para receber novos pedidos.

– **Funcionamento da Função:**

Atualização do Estado: O estado do *receptionist* é atualizado para WAIT_FOR_REQUEST, indicando que ele está a aguardar um pedido.

```
// TODO insert your code here
// Receptionist updates state
sh->fSt.st.receptionistStat = WAIT_FOR_REQUEST;
saveState(nFic,&(sh->fSt));
```

Esperar por um Pedido do *Group*: semDown(semgid, sh->receptionistReq): O *receptionist* espera por um sinal (receptionistReq) que indica um novo pedido de um *group*.

```
// TODO insert your code here
// Receptionist waits for request from group
if (semDown (semgid, sh->receptionistReq) == -1) {
    perror ("error on the down operation for receptionistReq semaphore (RT)");
    exit (EXIT_FAILURE);
}
```

Leitura e Armazenamento do Pedido: ret = sh->fSt.receptionistRequest: O pedido atual é armazenado na variável ret. Este pedido tanto pode ser do tipo TABLEREQ ou BILLREQ dependendo se o *group* está à espera de lhe ser atribuída uma mesa ou de efetuar o pagamento.

```
// TODO insert your code here
// Receptionist reads request
ret = sh->fSt.receptionistRequest;
```

Sinalização de disponibilidade: Após a leitura do pedido, o receptionist sai da região crítica. semUp(semgid, sh->receptionistRequestPossible): Sinaliza que está pronto para receber outro pedido.

```
// TODO insert your code here
// Receptionist signals availability for new request
if (semUp (semgid, sh->receptionistRequestPossible) == -1) {
    perror ("error on the up operation for receptionistRequestPossible semaphore (RT)");
    exit (EXIT_FAILURE);
}
```

Retorno da Função: A função retorna o pedido (ret), que contém as informações do pedido feito pelo *group*.

4.3.4 Função provideTableOrWaitingRoom()

A função provideTableOrWaitingRoom() é responsável por decidir se um group identificado pelo ID n, será alocado a uma mesa ou se deverá aguardar na sala de espera.

- **Propósito da Função:**

Decisão de Mesa ou Espera: Determinar se o *group* pode ser alocado a uma mesa imediatamente ou se deve esperar na sala de espera.

Atualização de Estados: Atualizar o estado do *receptionist* e do *group* com base na decisão de alocação de mesa.

Sinalização para o *Group*: Informar o *group* sobre a decisão, seja para ocupar uma mesa ou para esperar.

- **Funcionamento da Função:**

Atualização do Estado do *Receptionist*: O estado do *receptionist* é atualizado para ASSIGNTABLE, indicando que está no processo de alocação de mesa.

Decidir Mesa ou Espera: int tableId = decideTableOrWait(n): A função decideTableOrWait é chamada para determinar se há uma mesa disponível para o group n.

Alocação de Mesa ou Espera: Se tableId != -1, uma mesa está disponível. O *group* é então alocado à mesa (sh->fSt.assignedTable[n] = tableId) e o seu estado é atualizado para ATTABLE. O *group* é informado que pode prosseguir para a mesa. Se não houver mesa disponível (tableId == -1), o *group* deve esperar.

Atualização do Estado do *Receptionist*: O estado do *group* é atualizado para WAIT e o contador de *groups* à espera (sh->fSt.groupsWaiting) é incrementado.

```
// TODO insert your code here
// Receptionist updates state
sh->fSt.st.receptionistStat = ASSIGNTABLE;
saveState(nFic,&(sh->fSt));

int tableId = decideTableOrWait(n);
if (tableId != -1) { // Table available
    sh->fSt.assignedTable[n]=tableId; // Update table state
    groupRecord[n] = ATTABLE; // Update group state
    // Receptionist informs group that it may proceed
    if (semUp(semgid, sh->waitForTable[n]) == -1) {
        perror("error on the up operation for waitForTable semaphore (RT)");
        exit(EXIT_FAILURE);
    }
} else {
    // Group state is updated to WAIT
    sh->fSt.st.groupStat[tableId] = WAIT;
    sh->fSt.groupsWaiting++; // Update number of groups waiting
}
```

4.3.5 Função receivePayment()

A função receivePayment é responsável pelo processo de pagamento dos *groups* e pela gestão subsequente das mesas e dos *groups* em espera.

- **Propósito da Função:**

Receber Pagamento: Atualizar o estado do *receptionist* para indicar que ele está a processar um pagamento.

Liberar Mesa e Gerenciar Groups em Espera: Após receber o pagamento, verificar se há *groups* à espera e indicar uma mesa recém-libertada a um desses *groups*.

- **Funcionamento da Função:**

Atualização do Estado do *receptionist*: O estado do *receptionist* é atualizado para RECVPAY.

Sinalização de Mesa Liberada: semUp(semgid, sh->tableDone[sh>fSt.assignedTable[n]]): Notifica que a mesa atribuída ao *group* n está agora livre.

Atualização do Registo do Group: groupRecord[n] = DONE: Sinaliza o *group* n como concluído.

```
// TODO insert your code here
// Receptionist updates state
sh->fSt.st.receptionistStat = RECVPAY;
saveState(nFic, &sh->fSt);

// Receptionist receives payment
if (semUp (semgid, sh->tableDone[sh->fSt.assignedTable[n]]) == -1) {
    perror ("error on the down operation for semaphore access (WT)");
    exit (EXIT_FAILURE);
}
groupRecord[n] = DONE; // Update group state
```

Libertar Mesa: sh->fSt.assignedTable[n] = -1: Libertar a mesa que estava atribuída ao *group* n.

Gestão de *Groups* em Espera: Se houver *groups* a aguardar (sh>fSt.groupsWaiting > 0), a função decideNextGroup é chamada para identificar o próximo *group* a ser alocado. Nesse caso a função provideTableOrWaitingRoom(next_group) atribuirá uma mesa para o próximo *group* ou colocá-lo-á à espera. A seguir decrementa-se o contador de *groups* que estão à espera de mesa(sh->fSt.groupsWaiting--).

```
// TODO insert your code here
// Table is now vacant
sh->fSt.assignedTable[n]=-1;

// Receptionist checks if table that just became vacant should be occupied
if(sh->fSt.groupsWaiting>0){ // Check if there are waiting groups
    int next_group=decideNextGroup(); // Select next group
    provideTableOrWaitingRoom (next_group); // Provide table or waiting room
    sh->fSt.groupsWaiting--; // Update number of groups waiting
}
```

4.4 SemSharedMemGroup.c

4.4.1 Função checkInAtReception()

Esta função tem como objetivo tratar do check-in do *group*, na receção, para pedirem uma mesa ao receptionist.

- **Propósito da Função:**

Solicitar uma Mesa: Quando um *group* chega, ele deve pedir uma mesa na receção.

Aguardar Atribuição de Mesa: O *group* aguarda até que uma mesa lhe seja atribuída.

- **Funcionamento da Função**

Esperar pela Disponibilidade do *Receptionist*: semDown(semgid, sh->receptionistRequestPossible): O *group* espera até que o *receptionist* esteja disponível para processar o seu pedido.

```
// TODO insert your code here
// Check if receptionist is available
if (semDown(semgid, sh->receptionistRequestPossible) == -1) {
    perror("error on the down operation for receptionistRequestPossible semaphore (CT)");
    exit(EXIT_FAILURE);
}
```

Atualização do Estado do *Group*: O estado do *group* é atualizado para ATRECEPTION, indicando que está na receção a aguardar uma mesa. saveState(nFic, &sh->fSt): Guarda o estado atualizado do *group*.

Solicitação de Mesa: O *group* sinaliza ao *receptionist* que está a pedir uma mesa (TABLEREQ), sendo-lhe associado um id (reqGroup = id).

```
// TODO insert your code here
// Group updates state
sh->fSt.st.groupStat[id] = ATRECEPTION;
saveState(nFic, &sh->fSt);

// Group asks for a table
sh->fSt.receptionistRequest.reqType = TABLEREQ;
sh->fSt.receptionistRequest.reqGroup = id;

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

Sinalização ao Receptionist: semUp(semgid, sh->receptionistReq): O *group* sinaliza o *receptionist* que fez um pedido.

Aguardar Atribuição de Mesa: semDown(semgid, sh->waitForTable[id]): O *group* aguarda até que uma mesa lhe seja atribuída.

```
// TODO insert your code here
// Signal receptionist of the request
if (semUp(semgid, sh->receptionistReq) == -1) {
    perror("error on the up operation for receptionistReq semaphore (CT)");
    exit(EXIT_FAILURE);
}

// Group may have to wait for a table in this method.
if (semDown(semgid, sh->waitForTable[id]) == -1) {
    perror("error on the down operation for waitForTable semaphore (CT)");
    exit(EXIT_FAILURE);
}
```

4.4.2 Função orderFood()

A função orderFood é responsável pela ação de um *group* realizar um pedido de comida ao *receptionist*.

- **Propósito da Função:**

Fazer Pedido de Comida: Atualizar o estado do *group* para demonstrar que pretende realizar um pedido de comida ao *receptionist*.

Aguardar Disponibilidade do *Receptionist*: O *group* espera até que este esteja disponível para processar o seu pedido.

Aguardar Confirmação do Pedido: O *group* aguarda até que o *receptionist* confirme que recebeu o pedido.

- **Funcionamento da Função**

Esperar pela Disponibilidade do *Waiter*: semDown(semgid, sh->waiterRequestPossible): O *group* espera até que o *waiter* esteja disponível para receber pedidos.

```
// TODO insert your code here
// Check if waiter is available
if (semDown(semgid, sh->waiterRequestPossible) == -1) {
    perror("error on the down operation for waiterRequestPossible semaphore (CT)");
    exit(EXIT_FAILURE);
}
```

Atualização do Estado do Group: O estado do *group* é atualizado para FOOD_REQUEST, indicando que está a fazer um pedido de comida. saveState(nFic, &sh->fSt): Guarda o estado atualizado do *group*.

Solicitação de Comida: O *group* sinaliza o *waiter* que este pretende realizar um pedido comida (FOODREQ) e é lhe atribuído um id (reqGroup = id).

```
// TODO insert your code here
// Group updates state
sh->fSt.st.groupStat[id] = FOOD_REQUEST;
saveState(nFic, &sh->fSt);

// Group requests food to the waiter
sh->fSt.waiterRequest.reqType = FOODREQ;
sh->fSt.waiterRequest.reqGroup = id;
```

Sinalização de pedido feito: semUp(semgid, sh->waiterRequest): O *group* sinaliza o *waiter* que fez um pedido de comida.

Aguardar Confirmação do Pedido: semDown(semgid, sh->requestReceived[sh->fSt.assignedTable[id]]): O *group* aguarda a confirmação do *waiter* de que o pedido foi recebido.

```
// TODO insert your code here
// Group warns waiter of the request
if (semUp(semgid, sh->waiterRequest) == -1) {
    perror("error on the up operation for waiterRequest semaphore (CT)");
    exit(EXIT_FAILURE);
}

// Group wait for the waiter to receive the request
if (semDown(semgid, sh->requestReceived[sh->fSt.assignedTable[id]]) == -1) {
    perror("error on the down operation for requestReceived semaphore (CT)");
    exit(EXIT_FAILURE);
}
```

4.4.3 Função waitFood()

Esta função retrata a espera do *group* pela chegada da comida e quando esta lhe for entregue, irá começar a comer.

– **Propósito da Função:**

Aguardar Chegada da Comida: O *group* aguarda até que a comida chegue à mesa.

Iniciar a refeição: O *group* começa a comer, após a sua comida ser entregue.

– **Funcionamento da Função**

Atualização e Gravação do Estado para Espera da Comida: O estado do *group* é alterado para WAIT_FOR_FOOD, indicando que está à espera da comida. saveState(nFic, &sh->fSt): Guarda o estado atualizado do *group*.

```
// TODO insert your code here
// Group updates state
sh->fSt.st.groupStat[id] = WAIT_FOR_FOOD;
saveState(nFic, &sh->fSt);
```

Espera pela comida: semDown(semgid, sh->foodArrived[sh->fSt.assignedTable[id]]): O *group* espera até que a comida chegue.

```
// TODO insert your code here
// Group waits until food arrives.
if (semDown(semgid, sh->foodArrived[sh->fSt.assignedTable[id]]) == -1){
    perror("error on the down operation for foodArrived semaphore (CT)");
    exit(EXIT_FAILURE);
}
```

Atualização de Estado após chegada da comida: O estado do *group* é alterado para EAT, indicando que está a comer. saveState(nFic, &sh->fSt): Guarda o novo estado do *group*.

```
// TODO insert your code here
// It should also update state after food arrives.
sh->fSt.st.groupStat[id] = EAT;
saveState(nFic, &sh->fSt);
```

4.4.4 Função checkOutAtReception()

A função checkOutAtReception representa a ação de um group efetuar o pagamento na receção.

– **Propósito da Função:**

Enviar Pedido de Pagamento ao *Receptionist*: O *group* realiza um pedido de pagamento ao *receptionist*.

Aguardar Confirmação de Pagamento: O *group* espera até que o *receptionist* confirme o pagamento.

Sair do restaurante: Após realizar o pagamento, o *group* sai do restaurante.

– Funcionamento da Função

Espera pela Disponibilidade do Receptionist: semDown(semgid, sh->receptionistRequestPossible): O *group* aguarda até que o *receptionist* esteja disponível.

```
// TODO insert your code here
// Check if receptionist is available
if (semDown(semgid, sh->receptionistRequestPossible) == -1) {
    perror("error on the down operation for receptionistRequestPossible semaphore (CT)");
    exit(EXIT_FAILURE);
}
```

Atualização do estado do Group: O *group* atualiza o seu estado para CHECKOUT. saveState(nFic, &sh->fSt): Guarda o estado atualizado do *group*.

Envio do Pedido de Pagamento: O group envia um pedido de pagamento (BILLREQ) ao *receptionist*.

```
// TODO insert your code here
// Group updates state
sh->fSt.st.groupStat[id] = CHECKOUT;
saveState(nFic, &sh->fSt);

// Group sends a payment request to the receptionist
sh->fSt.receptionistRequest.reqType = BILLREQ;
sh->fSt.receptionistRequest.reqGroup = id;
```

Sinaliza *receptionist* e aguarda confirmação de pagamento: semUp(semgid, sh->receptionistReq): Sinaliza o pedido ao *receptionist*. semDown(semgid, sh->tableDone[sh->fSt.assignedTable[id]]): O *group* espera pela confirmação de pagamento.

```
// TODO insert your code here
// Signal receptionist of the request
if (semUp(semgid, sh->receptionistReq) == -1) {
    perror("error on the down operation for requestReceived semaphore (CT)");
    exit(EXIT_FAILURE);
}
// Group waits for receptionist to acknowledge payment.
if (semDown(semgid, sh->tableDone[sh->fSt.assignedTable[id]]) == -1) {
    perror("error on the down operation for tableDone semaphore (CT)");
    exit(EXIT_FAILURE);
}
```

Atualização do Estado para Saída: O *group* atualiza o seu estado *para LEAVING.* saveState(nFic, &sh->fSt): Guarda o novo estado do *group*.

```
// TODO insert your code here
// Group should update its state to LEAVING, after acknowledge.
sh->fSt.st.groupStat[id] = LEAVING;
saveState(nFic, &sh->fSt);
```

5. Resultados

Nesta secção do relatório vamos inserir os resultados que obtivemos ao correr a linha de comando “./run.sh 3”.

5.1 Run n.º1

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

 CH WT RC  G00 G01 G02 G03 G04  gWT T00 T01 T02 T03 T04
  0  0  0    1   1   1   1   1    0   .   .   .   .   .
  0  0  0    1   1   1   1   1    0   .   .   .   .   .
  0  0  0    1   1   1   1   1    0   .   .   .   .   .
  0  0  0    1   2   1   1   1    0   .   .   .   .   .
  0  0  1    1   2   1   1   1    0   .   .   .   .   .
  0  0  1    1   2   2   1   1    0   .   0   .   .   .
  0  0  0    1   2   2   1   1    0   .   0   .   .   .
  0  0  0    1   3   2   1   1    0   .   0   .   .   .
  0  0  1    1   3   2   1   1    0   .   0   .   .   .
  0  0  0    1   3   2   1   1    0   .   0   0   .   .
  0  0  0    1   3   2   1   1    0   .   0   0   .   .
  0  1  0    1   3   2   1   1    0   .   0   0   .   .
  0  1  0    1   3   3   1   1    0   .   0   0   .   .
  0  1  0    1   4   3   1   1    0   .   0   0   .   .
  1  1  0    1   4   3   1   1    0   .   0   0   .   .
  1  0  0    1   4   3   1   1    0   .   0   0   .   .
  1  0  0    1   4   3   1   1    0   .   0   0   .   .
  1  1  0    1   4   3   1   1    0   .   0   0   .   .
  1  1  0    1   4   4   1   1    0   .   0   0   .   .
  0  1  0    1   4   4   1   1    0   .   0   0   .   .
  1  1  0    1   4   4   1   1    0   .   0   0   .   .
  1  0  0    1   4   4   1   1    0   .   0   0   .   .
  1  0  0    1   4   4   1   1    0   .   0   0   .   .
  1  2  0    1   4   4   1   1    0   .   0   0   .   .
  1  0  0    1   4   4   1   1    0   .   0   0   .   .
  1  0  0    1   5   4   1   1    0   .   0   0   .   .
  0  0  0    1   5   4   1   1    0   .   0   0   .   .
  0  0  0    1   5   4   1   1    0   .   0   0   .   .
  0  2  0    1   5   4   1   1    0   .   0   0   .   .
  0  0  0    1   5   4   1   1    0   .   0   0   .   .
  0  0  0    1   5   5   1   1    0   .   0   0   .   .
  0  0  0    1   5   5   2   1    0   .   0   0   .   .
  0  0  1    1   5   5   2   1    0   .   0   0   .   .
  0  0  0    1   5   5   2   1    0   .   0   0   0   .
  0  0  0    1   5   5   3   1    0   .   0   0   0   .
  0  0  0    1   5   5   3   1    0   .   0   0   0   .
  0  1  0    1   5   5   3   1    0   .   0   0   0   .
  0  1  0    1   5   5   4   1    0   .   0   0   0   .
  1  1  0    1   5   5   4   1    0   .   0   0   0   .
  1  0  0    1   5   5   4   1    0   .   0   0   0   .
  0  0  0    1   5   5   4   1    0   .   0   0   0   .
  0  0  0    1   5   5   4   1    0   .   0   0   0   .
  0  2  0    1   5   5   4   1    0   .   0   0   0   .
  0  0  0    1   5   5   4   1    0   .   0   0   0   .
  0  0  0    1   5   5   5   1    0   .   0   0   0   .
  0  0  0    1   5   5   5   2    0   .   0   0   0   .
  0  0  1    1   5   5   5   2    0   .   0   0   0   .
  0  0  0    1   5   5   5   2    0   .   0   0   0   0
  0  0  0    1   5   5   5   3    0   .   0   0   0   0
  0  0  0    1   5   5   5   3    0   .   0   0   0   0
  0  1  0    1   5   5   5   3    0   .   0   0   0   0
  0  1  0    1   5   5   5   4    0   .   0   0   0   0
  1  1  0    1   5   5   5   4    0   .   0   0   0   0
  1  0  0    1   5   5   5   4    0   .   0   0   0   0
  0  0  0    1   5   5   5   4    0   .   0   0   0   0
  0  0  0    1   5   5   5   4    0   .   0   0   0   0
  0  2  0    1   5   5   5   4    0   .   0   0   0   0
  0  0  0    1   5   5   5   4    0   .   0   0   0   0
  0  0  0    1   5   5   5   5    0   .   0   0   0   0
  0  0  0    2   5   5   5   5    0   .   0   0   0   0
  0  0  1    2   5   5   5   5    0   .   0   0   0   0
  0  0  0    2   5   5   5   5    0   0   0   0   0   0
  0  0  0    3   5   5   5   5    0   0   0   0   0   0
  0  0  0    3   5   5   5   5    0   0   0   0   0   0
  0  1  0    3   5   5   5   5    0   0   0   0   0   0
  0  1  0    4   5   5   5   5    0   0   0   0   0   0
  1  1  0    4   5   5   5   5    0   0   0   0   0   0
  1  0  0    4   5   5   5   5    0   0   0   0   0   0
  0  0  0    4   5   5   5   5    0   0   0   0   0   0
  0  0  0    4   5   5   5   5    0   0   0   0   0   0
  0  2  0    4   5   5   5   5    0   0   0   0   0   0
  0  2  0    5   5   5   5   5    0   0   0   0   0   0
  0  2  0    5   5   5   6   5    0   0   0   0   0   0
  0  2  2    5   5   5   6   5    0   0   0   0   0   0
  0  2  0    5   5   5   6   5    0   0   0   0   .   0
  0  2  0    5   5   5   7   5    0   0   0   0   .   0
  0  2  0    5   5   5   7   6    0   0   0   0   .   0
  0  2  2    5   5   5   7   6    0   0   0   0   .   0
  0  2  0    5   5   5   7   6    0   0   0   0   .   .
  0  2  0    5   5   5   7   7    0   0   0   0   .   .
  0  2  0    6   5   5   7   7    0   0   0   0   .   .
  0  2  2    6   5   5   7   7    0   0   0   0   .   .
  0  2  0    6   5   5   7   7    0   .   0   0   .   .
  0  2  0    7   5   5   7   7    0   .   0   0   .   .
  0  2  0    7   5   6   7   7    0   .   0   0   .   .
  0  2  2    7   5   6   7   7    0   .   0   0   .   .
  0  2  0    7   5   6   7   7    0   .   0   .   .   .
  0  2  0    7   5   7   7   7    0   .   0   .   .   .
  0  2  0    7   6   7   7   7    0   .   0   .   .   .
  0  2  2    7   6   7   7   7    0   .   0   .   .   .
  0  2  2    7   7   7   7   7    0   .   .   .   .   .
```

5.2 Run n.º2

```
Run n.º 2
                                Restaurant - Description of the internal state

 CH WT RC  G00 G01 G02 G03 G04  gWT T00 T01 T02 T03 T04
  0  0  0    1   1   1   1   1    0   .   .   .   .   .
  0  0  0    1   1   1   1   1    0   .   .   .   .   .
  0  0  0    1   1   1   1   1    0   .   .   .   .   .
  0  0  0    1   2   1   1   1    0   .   .   .   .   .
  0  0  1    1   2   1   1   1    0   .   .   .   .   .
  0  0  1    1   2   2   1   1    0   .   0   .   .   .
  0  0  0    1   2   2   1   1    0   .   0   .   .   .
  0  0  0    1   3   2   1   1    0   .   0   .   .   .
  0  0  1    1   3   2   1   1    0   .   0   .   .   .
  0  0  0    1   3   2   1   1    0   .   0   0   .   .
  0  0  0    1   3   2   1   1    0   .   0   0   .   .
  0  1  0    1   3   2   1   1    0   .   0   0   .   .
  0  1  0    1   3   3   1   1    0   .   0   0   .   .
  0  1  0    1   4   3   1   1    0   .   0   0   .   .
  1  1  0    1   4   3   1   1    0   .   0   0   .   .
  1  0  0    1   4   3   1   1    0   .   0   0   .   .
  1  0  0    1   4   3   1   1    0   .   0   0   .   .
  1  1  0    1   4   3   1   1    0   .   0   0   .   .
  1  1  0    1   4   4   1   1    0   .   0   0   .   .
  0  1  0    1   4   4   1   1    0   .   0   0   .   .
  1  1  0    1   4   4   1   1    0   .   0   0   .   .
  1  0  0    1   4   4   1   1    0   .   0   0   .   .
  1  0  0    1   4   4   1   1    0   .   0   0   .   .
  1  2  0    1   4   4   1   1    0   .   0   0   .   .
  1  0  0    1   4   4   1   1    0   .   0   0   .   .
  1  0  0    1   5   4   1   1    0   .   0   0   .   .
  0  0  0    1   5   4   1   1    0   .   0   0   .   .
  0  0  0    1   5   4   1   1    0   .   0   0   .   .
  0  2  0    1   5   4   1   1    0   .   0   0   .   .
  0  0  0    1   5   4   1   1    0   .   0   0   .   .
  0  0  0    1   5   5   1   1    0   .   0   0   .   .
  0  0  0    1   5   5   2   1    0   .   0   0   .   .
  0  0  1    1   5   5   2   1    0   .   0   0   .   .
  0  0  0    1   5   5   2   1    0   .   0   0   0   .
  0  0  0    1   5   5   3   1    0   .   0   0   0   .
  0  0  0    1   5   5   3   1    0   .   0   0   0   .
  0  1  0    1   5   5   3   1    0   .   0   0   0   .
  0  1  0    1   5   5   4   1    0   .   0   0   0   .
  1  1  0    1   5   5   4   1    0   .   0   0   0   .
  1  0  0    1   5   5   4   1    0   .   0   0   0   .
  0  0  0    1   5   5   4   1    0   .   0   0   0   .
  0  0  0    1   5   5   4   1    0   .   0   0   0   .
  0  2  0    1   5   5   4   1    0   .   0   0   0   .
  0  0  0    1   5   5   4   1    0   .   0   0   0   .
  0  0  0    1   5   5   5   1    0   .   0   0   0   .
  0  0  0    1   5   5   5   2    0   .   0   0   0   .
  0  0  1    1   5   5   5   2    0   .   0   0   0   .
  0  0  0    1   5   5   5   2    0   .   0   0   0   0
  0  0  0    1   5   5   5   3    0   .   0   0   0   0
  0  0  0    1   5   5   5   3    0   .   0   0   0   0
  0  1  0    1   5   5   5   3    0   .   0   0   0   0
  0  1  0    1   5   5   5   4    0   .   0   0   0   0
  1  1  0    1   5   5   5   4    0   .   0   0   0   0
  1  0  0    1   5   5   5   4    0   .   0   0   0   0
  0  0  0    1   5   5   5   4    0   .   0   0   0   0
  0  0  0    1   5   5   5   4    0   .   0   0   0   0
  0  2  0    1   5   5   5   4    0   .   0   0   0   0
  0  0  0    1   5   5   5   4    0   .   0   0   0   0
  0  0  0    1   5   5   5   5    0   .   0   0   0   0
  0  0  0    2   5   5   5   5    0   .   0   0   0   0
  0  0  1    2   5   5   5   5    0   .   0   0   0   0
  0  0  0    2   5   5   5   5    0   0   0   0   0   0
  0  0  0    3   5   5   5   5    0   0   0   0   0   0
  0  0  0    3   5   5   5   5    0   0   0   0   0   0
  0  1  0    3   5   5   5   5    0   0   0   0   0   0
  0  1  0    4   5   5   5   5    0   0   0   0   0   0
  1  1  0    4   5   5   5   5    0   0   0   0   0   0
  1  0  0    4   5   5   5   5    0   0   0   0   0   0
  0  0  0    4   5   5   5   5    0   0   0   0   0   0
  0  0  0    4   5   5   5   5    0   0   0   0   0   0
  0  2  0    4   5   5   5   5    0   0   0   0   0   0
  0  2  0    5   5   5   5   5    0   0   0   0   0   0
  0  2  0    5   5   5   6   5    0   0   0   0   0   0
  0  2  2    5   5   5   6   5    0   0   0   0   0   0
  0  2  0    5   5   5   6   5    0   0   0   0   .   0
  0  2  0    5   5   5   7   5    0   0   0   0   .   0
  0  2  0    5   5   5   7   6    0   0   0   0   .   0
  0  2  2    5   5   5   7   6    0   0   0   0   .   0
  0  2  0    5   5   5   7   6    0   0   0   0   .   .
  0  2  0    5   5   5   7   7    0   0   0   0   .   .
  0  2  0    6   5   5   7   7    0   0   0   0   .   .
  0  2  2    6   5   5   7   7    0   0   0   0   .   .
  0  2  0    6   5   5   7   7    0   .   0   0   .   .
  0  2  0    7   5   5   7   7    0   .   0   0   .   .
  0  2  0    7   5   6   7   7    0   .   0   0   .   .
  0  2  2    7   5   6   7   7    0   .   0   0   .   .
  0  2  0    7   5   6   7   7    0   .   0   .   .   .
  0  2  0    7   5   7   7   7    0   .   0   .   .   .
  0  2  0    7   6   7   7   7    0   .   0   .   .   .
  0  2  2    7   6   7   7   7    0   .   0   .   .   .
  0  2  2    7   7   7   7   7    0   .   .   .   .   .
```

5.3 Run n.º3

Run n.º 3

Restaurant - Description of the internal state

CH	WT	RC	G00	G01	G02	G03	G04	gWT	T00	T01	T02	T03	T04
0	0	0	1	1	1	1	1	0	.	.	.	.	.
0	0	0	1	1	1	1	1	0	.	.	.	.	.
0	0	0	1	1	1	1	1	0	.	.	.	.	.
0	0	0	1	2	1	1	1	0	.	.	.	.	.
0	0	1	1	2	1	1	1	0	.	.	.	.	.
0	0	1	1	2	2	1	1	0	.	0	.	.	.
0	0	0	1	2	2	1	1	0	.	0	.	.	.
0	0	0	1	3	2	1	1	0	.	0	.	.	.
0	0	1	1	3	2	1	1	0	.	0	.	.	.
0	0	0	1	3	2	1	1	0	.	0	0	.	.
0	0	0	1	3	2	1	1	0	.	0	0	.	.
0	1	0	1	3	2	1	1	0	.	0	0	.	.
0	1	0	1	3	3	1	1	0	.	0	0	.	.
0	1	0	1	4	3	1	1	0	.	0	0	.	.
1	1	0	1	4	3	1	1	0	.	0	0	.	.
1	0	0	1	4	3	1	1	0	.	0	0	.	.
1	0	0	1	4	3	1	1	0	.	0	0	.	.
1	1	0	1	4	3	1	1	0	.	0	0	.	.
1	1	0	1	4	4	1	1	0	.	0	0	.	.
0	1	0	1	4	4	1	1	0	.	0	0	.	.
1	1	0	1	4	4	1	1	0	.	0	0	.	.
1	0	0	1	4	4	1	1	0	.	0	0	.	.
1	0	0	1	4	4	1	1	0	.	0	0	.	.
1	2	0	1	4	4	1	1	0	.	0	0	.	.
1	0	0	1	4	4	1	1	0	.	0	0	.	.
1	0	0	1	5	4	1	1	0	.	0	0	.	.
0	0	0	1	5	4	1	1	0	.	0	0	.	.
0	0	0	1	5	4	1	1	0	.	0	0	.	.
0	2	0	1	5	4	1	1	0	.	0	0	.	.
0	0	0	1	5	4	1	1	0	.	0	0	.	.
0	0	0	1	5	5	1	1	0	.	0	0	.	.
0	0	0	1	5	5	2	1	0	.	0	0	.	.
0	0	1	1	5	5	2	1	0	.	0	0	.	.
0	0	0	1	5	5	2	1	0	.	0	0	0	.
0	0	0	1	5	5	3	1	0	.	0	0	0	.
0	0	0	1	5	5	3	1	0	.	0	0	0	.
0	1	0	1	5	5	3	1	0	.	0	0	0	.
0	1	0	1	5	5	4	1	0	.	0	0	0	.
1	1	0	1	5	5	4	1	0	.	0	0	0	.
1	0	0	1	5	5	4	1	0	.	0	0	0	.
0	0	0	1	5	5	4	1	0	.	0	0	0	.
0	0	0	1	5	5	4	1	0	.	0	0	0	.
0	2	0	1	5	5	4	1	0	.	0	0	0	.
0	0	0	1	5	5	4	1	0	.	0	0	0	.
0	0	0	1	5	5	5	1	0	.	0	0	0	.
0	0	0	1	5	5	5	2	0	.	0	0	0	.
0	0	1	1	5	5	5	2	0	.	0	0	0	.
0	0	0	1	5	5	5	2	0	.	0	0	0	0
0	0	0	1	5	5	5	3	0	.	0	0	0	0
0	0	0	1	5	5	5	3	0	.	0	0	0	0
0	1	0	1	5	5	5	3	0	.	0	0	0	0
0	1	0	1	5	5	5	4	0	.	0	0	0	0
1	1	0	1	5	5	5	4	0	.	0	0	0	0
1	0	0	1	5	5	5	4	0	.	0	0	0	0
0	0	0	1	5	5	5	4	0	.	0	0	0	0
0	0	0	1	5	5	5	4	0	.	0	0	0	0
0	2	0	1	5	5	5	4	0	.	0	0	0	0
0	0	0	1	5	5	5	4	0	.	0	0	0	0
0	0	0	1	5	5	5	5	0	.	0	0	0	0
0	0	0	2	5	5	5	5	0	.	0	0	0	0
0	0	1	2	5	5	5	5	0	.	0	0	0	0
0	0	0	2	5	5	5	5	0	0	0	0	0	0
0	0	0	3	5	5	5	5	0	0	0	0	0	0
0	0	0	3	5	5	5	5	0	0	0	0	0	0
0	1	0	3	5	5	5	5	0	0	0	0	0	0
0	1	0	4	5	5	5	5	0	0	0	0	0	0
1	1	0	4	5	5	5	5	0	0	0	0	0	0
1	0	0	4	5	5	5	5	0	0	0	0	0	0
0	0	0	4	5	5	5	5	0	0	0	0	0	0
0	0	0	4	5	5	5	5	0	0	0	0	0	0
0	2	0	4	5	5	5	5	0	0	0	0	0	0
0	2	0	5	5	5	5	5	0	0	0	0	0	0
0	2	0	5	5	5	6	5	0	0	0	0	0	0
0	2	2	5	5	5	6	5	0	0	0	0	0	0
0	2	0	5	5	5	6	5	0	0	0	0	.	0
0	2	0	5	5	5	7	5	0	0	0	0	.	0
0	2	0	5	5	5	7	6	0	0	0	0	.	0
0	2	2	5	5	5	7	6	0	0	0	0	.	0
0	2	0	5	5	5	7	6	0	0	0	0	.	.
0	2	0	5	5	5	7	7	0	0	0	0	.	.
0	2	0	6	5	5	7	7	0	0	0	0	.	.
0	2	2	6	5	5	7	7	0	0	0	0	.	.
0	2	0	6	5	5	7	7	0	.	0	0	.	.
0	2	0	7	5	5	7	7	0	.	0	0	.	.
0	2	0	7	5	6	7	7	0	.	0	0	.	.
0	2	2	7	5	6	7	7	0	.	0	0	.	.
0	2	0	7	5	6	7	7	0	.	0	.	.	.
0	2	0	7	5	7	7	7	0	.	0	.	.	.
0	2	0	7	6	7	7	7	0	.	0	.	.	.
0	2	2	7	6	7	7	7	0	.	0	.	.	.
0	2	2	7	7	7	7	7	0	.	.	.	.	.

Conclusão

Este trabalho foi extremamente importante para percebermos a utilidade e aplicação da utilização de semáforos e da memória partilhada na sincronização de diversos processos, uma vez que sem este recurso o programa ficaria extremamente condicionado e sujeito a falhas.

Um dos maiores desafios foi gerir a concorrência entre as múltiplas entidades. Isso foi superado com o uso cuidadoso de semáforos para controlar o acesso a recursos partilhados e áreas críticas.

O sucesso deste projeto não reside apenas na sua execução técnica, mas também na capacidade de simular de forma realista e eficiente o ambiente dinâmico de um restaurante, abrindo caminho para futuras melhorias e adaptações em sistemas similares.

Dito isto, obtemos os resultados esperados, atingindo assim todos os objetivos propostos e estamos confiantes no seu bom desempenho.