

Sistemas Operativos

Prof. Nuno Lau



RELATÓRIO – Trabalho 2

Restaurante

David Filipe Ferreira Amorim, nº 112610, P5
Francisca Ferreira Pedro Silva, nº 112841, P5

Licenciatura em Engenharia Informática

Aveiro
2023/2024

Índice

Introdução	1
1 – Processos	2
1.1 – Grupos (Groups)	2
1.2 – Rececionista (Receptionist)	5
1.3 – Empregado de mesa (Waiter).....	7
1.4 – Chef.....	9
2 – Testagem	10
Conclusão	13

Introdução

Enquanto estudantes da Licenciatura em Engenharia Informática na Universidade de Aveiro, no âmbito da cadeira Sistemas Operativos, tivemos como objetivo deste trabalho a compreensão dos mecanismos associados à execução e sincronização de processos e threads.

Deste modo, ao longo do presente relatório iremos explicar detalhadamente a abordagem que tomamos para a realização/resolução do problema e testagem para validação da solução que compõem o trabalho.

1 – Processos

Aquando da simulação deste restaurante, os grupos, o rececionista, o empregado de mesa e o cozinheiro são processos independentes, pelo que a sua sincronização é feita através de semáforos e de memória partilhada.

Nome do Semáforo	quem faz Down?	Quando?	quem faz Up?	Quando?
mutex	todos	É necessário fazer alterações na memória partilhada	todos	Terminam de fazer alterações na memória partilhada
receptionistReq	receptionist	Espera que o group chegue ou faça o checkout	group	Chega ou faz o checkout
receptionistRequestPossible	group	Chega ao restaurante ou pretende realizar o checkout	receptionist	Está disponível para atender o group
waiterRequest	waiter	Espera que a comida esteja pronta ou que o group faça o pedido	chef/group	O comida está pronta/ Acabou de encomendar o pedido
waiterRequestPossible	chef/group	Está a processar um pedido/ Está a encomendar o pedido	waiter	Está disponível para atender o chef/group
WaitOrder	chef	Está à espera de um pedido	waiter	Entrega um pedido ao chefe
orderReceived	waiter	Espera que o chef confirme a receção do pedido	chef	Confirma a receção de um pedido
waitForTable[MAXGROUPS]	group	Espera por uma mesa disponível	receptionist	Há uma mesa disponível para ser ocupada
requestReceived[NUMTABLES]	group	Espera que o waiter saiba que o pedido está a ser tratado pelo chef	waiter	Sabe que o chef está a tratar do pedido
foodArrived[NUMTABLES]	group	Espera que a comida chegue	waiter	Entrega a comida ao grupo
tableDone[NUMTABLES]	group	Está a fazer o checkout na receção	receptionist	O pagamento foi realizado com sucesso

1.1 – Grupos (Groups)

Este processo é constituído por seis operações representadas pelas funções:

- goToRestaurant;
- checkInAtReception;
- orderFood;
- waitFood;
- eat;
- checkOutAtReception.

Neste trabalho foi solicitada a alteração de quatro destas funções.

Na função `checkInAtReception` o grupo chega ao Restaurante onde pretende jantar e dirige-se ao rececionista, aguarda pela disponibilidade do mesmo (1), altera o estado para `ATRECEPTION` (2) e solicita uma mesa (3), o rececionista processa o pedido (4), enquanto o grupo aguarda até que se encontre alguma mesa disponível (5).

```

1 static void checkInAtReception(int id) {
2     // TODO insert your code here
3     if (semDown(semgid, RECEPTIONISTREQUESTPOSSIBLE) == -1) { (1)
4         perror("error on the down operation for semaphore access (GR-RECEPTIONISTREQUESTPOSSIBLE)");
5         exit(EXIT_FAILURE);
6     }
7
8     if (semDown(semgid, sh->mutex) == -1) { /* enter critical region */
9         perror("error on the down operation for semaphore access (GR)");
10        exit(EXIT_FAILURE);
11    }
12
13    // TODO insert your code here
14    sh->fSt.groupStat[id] = ATRECEPTION; (2)
15    sh->fSt.receptionistRequest.reqGroup = id; (3)
16    sh->fSt.receptionistRequest.reqType = TABLEREQ;
17    saveState(nFic, &sh->fSt);
18
19    if (semUp(semgid, sh->mutex) == -1) { /* exit critical region */
20        perror("error on the up operation for semaphore access (GR)");
21        exit(EXIT_FAILURE);
22    }
23
24    // TODO insert your code here
25
26    if (semUp(semgid, RECEPTIONISTREQ) == -1) { (4)
27        perror("error on the up operation for semaphore access (GR-RECEPTIONISTREQ)");
28        exit(EXIT_FAILURE);
29    }
30
31    if (semDown(semgid, sh->waitForTable[id]) == -1) { (5)
32        perror("error on the down operation for semaphore access (GR-WAITFORTABLE)");
33        exit(EXIT_FAILURE);
34    }
35 }

```

Na função `orderFood` o grupo aguarda pela disponibilidade do empregado de mesa (1), altera o estado (2), faz um pedido (3) e aguarda que o pedido seja recebido (4).

```

1 static void orderFood(int id) {
2     // TODO insert your code here
3     if (semDown(semgid, WAITERREQUESTPOSSIBLE) == -1) { (1)
4         perror("error on the down operation for semaphore access (GR-WAITERREQUESTPOSSIBLE)");
5         exit(EXIT_FAILURE);
6     }
7
8     if (semDown(semgid, sh->mutex) == -1) { /* enter critical region */
9         perror("error on the down operation for semaphore access (GR)");
10        exit(EXIT_FAILURE);
11    }
12
13    // TODO insert your code here
14    sh->fSt.groupStat[id] = FOOD_REQUEST; (2)
15    sh->fSt.waiterRequest.reqGroup = id; (3)
16    sh->fSt.waiterRequest.reqType = FOODREQ;
17    saveState(nFic, &sh->fSt);
18
19    if (semUp(semgid, sh->mutex) == -1) { /* exit critical region */
20        perror("error on the up operation for semaphore access (GR)");
21        exit(EXIT_FAILURE);
22    }
23
24    // TODO insert your code here
25
26    if (semUp(semgid, WAITERREQUEST) == -1) { (4)
27        perror("error on the up operation for semaphore access (GR-WAITERREQUEST)");
28        exit(EXIT_FAILURE);
29    }
30
31    if (semDown(semgid, sh->requestReceived[sh->fSt.assignedTable[id]]) == -1) { (5)
32        perror("error on the down operation for semaphore access (GR-REQUESTRECEIVED)");
33        exit(EXIT_FAILURE);
34    }
35 }

```

Na função `waitFood` altera o estado para `WAIT_FOR_FOOD` (1) e aguarda que a comida chega (2), após a chegada da mesma come e atualiza o seu estado para `EAT` (3).

```

1 static void waitFood(int id) {
2     if (semDown(semgid, sh->mutex) == -1) { /* enter critical region */
3         perror("error on the down operation for semaphore access (GR)");
4         exit(EXIT_FAILURE);
5     }
6
7     // TODO insert your code here
8     sh->fSt.st.groupStat[id] = WAIT_FOR_FOOD;
9     saveState(nFic, &sh->fSt);
10
11    if (semUp(semgid, sh->mutex) == -1) { /* enter critical region */
12        perror("error on the up operation for semaphore access (GR)");
13        exit(EXIT_FAILURE);
14    }
15
16    // TODO insert your code here
17    if (semDown(semgid, sh->foodArrived[sh->fSt.assignedTable[id]]) == -1) {
18        perror("error on the down operation for semaphore access (GR- FOODARRIVED)");
19        exit(EXIT_FAILURE);
20    }
21
22    if (semDown(semgid, sh->mutex) == -1) { /* enter critical region */
23        perror("error on the down operation for semaphore access (GR)");
24        exit(EXIT_FAILURE);
25    }
26
27    // TODO insert your code here
28    sh->fSt.st.groupStat[id] = EAT;
29    saveState(nFic, &sh->fSt);
30
31    if (semUp(semgid, sh->mutex) == -1) { /* enter critical region */
32        perror("error on the up operation for semaphore access (GR)");
33        exit(EXIT_FAILURE);
34    }
35 }

```

Na função `checkOutAtReception` o grupo aguarda pela disponibilidade do mesmo (1), altera o estado para `CHECKOUT` (2) e solicita a conta (3), o rececionista processa o pedido (4), o grupo aguarda a conclusão do checkout (5), sai e atualiza o seu estado para `LEAVING` (6).

```

1 static void checkOutAtReception(int id) {
2     // TODO insert your code here
3     if (semDown(semgid, RECEPTIONISTREQUESTPOSSIBLE) == -1) {
4         perror("error on the down operation for semaphore access (GR-RECEPTIONISTREQUESTPOSSIBLE)");
5         exit(EXIT_FAILURE);
6     }
7     if (semDown(semgid, sh->mutex) == -1) { /* enter critical region */
8         perror("error on the down operation for semaphore access (GR)");
9         exit(EXIT_FAILURE);
10    }
11    // TODO insert your code here
12    sh->fSt.st.groupStat[id] = CHECKOUT;
13    sh->fSt.receptionistRequest.reqGroup = id;
14    sh->fSt.receptionistRequest.reqType = BILLREQ;
15    saveState(nFic, &sh->fSt);
16    if (semUp(semgid, sh->mutex) == -1) { /* enter critical region */
17        perror("error on the up operation for semaphore access (GR)");
18        exit(EXIT_FAILURE);
19    }
20    // TODO insert your code here
21    if (semUp(semgid, RECEPTIONISTREQ) == -1) {
22        perror("error on the up operation for semaphore access (GR-RECEPTIONISTREQ)");
23        exit(EXIT_FAILURE);
24    }
25    if (semDown(semgid, sh->tableDone[sh->fSt.assignedTable[id]]) == -1) {
26        perror("error on the down operation for semaphore access (GR-TABLEDONE)");
27        exit(EXIT_FAILURE);
28    }
29    if (semDown(semgid, sh->mutex) == -1) { /* enter critical region */
30        perror("error on the down operation for semaphore access (GR)");
31        exit(EXIT_FAILURE);
32    }
33    // TODO insert your code here
34    sh->fSt.st.groupStat[id] = LEAVING;
35    saveState(nFic, &sh->fSt);
36    if (semUp(semgid, sh->mutex) == -1) { /* enter critical region */
37        perror("error on the up operation for semaphore access (GR)");
38        exit(EXIT_FAILURE);
39    }
40 }

```

1.2 – Rececionista (Receptionist)

O rececionista executa a operação `waitForGroup`, `provideTableOrWaitingRoom` e `receivePayment`.

A função `waitForGroup` atualiza o estado da função para `WAIT_FOR_REQUEST`, e lê o pedido e diz se é possível realizar um novo pedido.

```
1 static request waitForGroup() {
2     request ret;
3
4     if (semDown(semgid, sh->mutex) == -1) { /* enter critical region */
5         perror("error on the down operation for semaphore access (RC)");
6         exit(EXIT_FAILURE);
7     }
8
9     // TODO insert your code here
10    sh->fSt.receptionistStat = WAIT_FOR_REQUEST;
11    saveState(nFic, &sh->fSt);
12
13    if (semUp(semgid, sh->mutex) == -1) { /* exit critical region */
14        perror("error on the up operation for semaphore access (RC)");
15        exit(EXIT_FAILURE);
16    }
17
18    // TODO insert your code here
19    if (semDown(semgid, RECEPTIONISTREQ) == -1) {
20        perror("error on the down operation for semaphore access (RC-RECEPTIONISTREQ)");
21        exit(EXIT_FAILURE);
22    }
23
24    if (semDown(semgid, sh->mutex) == -1) { /* enter critical region */
25        perror("error on the down operation for semaphore access (RC)");
26        exit(EXIT_FAILURE);
27    }
28
29    // TODO insert your code here
30    ret = sh->fSt.receptionistRequest;
31
32    if (semUp(semgid, sh->mutex) == -1) { /* exit critical region */
33        perror("error on the up operation for semaphore access (RC)");
34        exit(EXIT_FAILURE);
35    }
36
37    // TODO insert your code here
38    if (semUp(semgid, RECEPTIONISTREQUESTPOSSIBLE) == -1) {
39        perror("error on the up operation for semaphore access (RC-RECEPTIONISTREQUESTPOSSIBLE)");
40        exit(EXIT_FAILURE);
41    }
42
43    return ret;
44 }
```

Por sua vez, a função `provideTableOrWaitingRoom`, à semelhança da função anterior, atualiza o estado da função para `ASSIGNTABLE` e decide qual a localização para onde o grupo se deverá dirigir. Esta decisão recorre à função `decideTableOrWait` e caso a mesa esteja disponível o rececionista deverá indicá-lo.


```

1 static void provideTableOrWaitingRoom(int n) {
2     if (semDown(semgid, sh->mutex) == -1) { /* enter critical region */
3         perror("error on the down operation for semaphore access (RC)");
4         exit(EXIT_FAILURE);
5     }
6
7     // TODO insert your code here
8     sh->fSt.st.receptionistStat = ASSIGNTABLE;
9
10    int table = decideTableOrWait(n);
11
12    if (table == -1) {
13        sh->fSt.groupsWaiting++;
14        groupRecord[n] = WAIT;
15    } else {
16        sh->fSt.assignedTable[n] = table;
17        groupRecord[n] = ATTABLE;
18        if (semUp(semgid, sh->waitForTable[n]) == -1) {
19            perror("error on the up operation for semaphore access (RC-WAITFORTABLE)");
20            exit(EXIT_FAILURE);
21        }
22    }
23
24    saveState(nFic, &sh->fSt);
25
26    if (semUp(semgid, sh->mutex) == -1) { /* exit critical region */
27        perror("error on the up operation for semaphore access (RC)");
28        exit(EXIT_FAILURE);
29    }
30 }

```

Aquando do pagamento, a função `receivePayment` atualiza o estado para `RECVPAY`, recebe o pagamento e indica que o pagamento foi realizado com sucesso e, caso existam grupos à espera indica que há uma mesa disponível.

```

1 static void receivePayment(int n) {
2     if (semDown(semgid, sh->mutex) == -1) { /* enter critical region */
3         perror("error on the down operation for semaphore access (RC)");
4         exit(EXIT_FAILURE);
5     }
6
7     // TODO insert your code here
8     sh->fSt.st.receptionistStat = RECVPAY;
9     groupRecord[n] = DONE;
10    if (semUp(semgid, sh->tableDone[sh->fSt.assignedTable[n]]) == -1) {
11        perror("error on the up operation for semaphore access (RC-TABLEDONE)");
12        exit(EXIT_FAILURE);
13    }
14
15    int next_group = decideNextGroup();
16
17    if (next_group != -1) {
18        groupRecord[next_group] = ATTABLE;
19        sh->fSt.assignedTable[next_group] = sh->fSt.assignedTable[n];
20        sh->fSt.assignedTable[n] = -1;
21        sh->fSt.groupsWaiting--;
22    }
23
24    saveState(nFic, &sh->fSt);
25
26    if (semUp(semgid, sh->mutex) == -1) { /* exit critical region */
27        perror("error on the up operation for semaphore access (RC)");
28        exit(EXIT_FAILURE);
29    }
30
31    // TODO insert your code here
32    if (next_group != -1) {
33        if (semUp(semgid, sh->waitForTable[next_group]) == -1) {
34            perror("error on the up operation for semaphore access (RC-WAITFORTABLE)");
35            exit(EXIT_FAILURE);
36        }
37    }

```


1.3 – Empregado de mesa (Waiter)

Este processo é constituído por duas operações representadas pelas funções:

- waitForClientOrChef
- informChef
- takeFoodToTable

A primeira função é muito semelhante à função waitForGroup do processo Rececionista.

```
1 static request waitForClientOrChef() {
2     request req;
3     if (semDown(semgid, sh->mutex) == -1) { /* enter critical region */
4         perror("error on the down operation for semaphore access (WT)");
5         exit(EXIT_FAILURE);
6     }
7
8     // TODO insert your code here
9     // Updates the waiter stat to wait for a request
10    sh->fSt.waiterStat = WAIT_FOR_REQUEST;
11    saveState(nFic, &sh->fSt);
12
13    if (semUp(semgid, sh->mutex) == -1) { /* exit critical region */
14        perror("error on the up operation for semaphore access (WT)");
15        exit(EXIT_FAILURE);
16    }
17
18    // TODO insert your code here
19    // Waits until the waiter receives a request
20    if (semDown(semgid, WAITERREQUEST) == -1) {
21        perror("error on the down operation for semaphore access (WT-WAITERREQUEST)");
22        exit(EXIT_FAILURE);
23    }
24
25    if (semDown(semgid, sh->mutex) == -1) { /* enter critical region */
26        perror("error on the down operation for semaphore access (WT)");
27        exit(EXIT_FAILURE);
28    }
29
30    // TODO insert your code here
31    req = sh->fSt.waiterRequest;
32
33    if (semUp(semgid, sh->mutex) == -1) { /* exit critical region */
34        perror("error on the up operation for semaphore access (WT)");
35        exit(EXIT_FAILURE);
36    }
37
38    // TODO insert your code here
39    if (semUp(semgid, WAITERREQUESTPOSSIBLE) == -1) {
40        perror("error on the up operation for semaphore access (WT-WAITERREQUESTPOSSIBLE)");
41        exit(EXIT_FAILURE);
42    }
43
44    return req;
45 }
```

A segunda função atualiza o estado para `INFORM_CHEF`, entrega um pedido ao chef, o empregado aguardar que o chef confirme a receção do pedido e verifica que o chef está a tratar do pedido.

```
1 static void informChef(int n) {
2     if (semDown(semgid, sh->mutex) == -1) { /* enter critical region */
3         perror("error on the down operation for semaphore access (WT)");
4         exit(EXIT_FAILURE);
5     }
6
7     // TODO insert your code here
8     sh->fSt.waiterStat = INFORM_CHEF;
9     sh->fSt.foodGroup = n;
10    sh->fSt.foodOrder = 1;
11    saveState(nFic, &sh->fSt);
12
13    if (semUp(semgid, sh->mutex) == -1) /* exit critical region */
14    {
15        perror("error on the up operation for semaphore access (WT)");
16        exit(EXIT_FAILURE);
17    }
18
19    // TODO insert your code here
20    // Wakes up the chef
21    if (semUp(semgid, WAITORDER) == -1) {
22        perror("error on the up operation for semaphore access (WT-WAITORDER)");
23        exit(EXIT_FAILURE);
24    }
25    if (semDown(semgid, ORDERRECEIVED) == -1) {
26        perror("error on the down operation for semaphore access (WT-ORDERRECEIVED)");
27        exit(EXIT_FAILURE);
28    }
29    if (semUp(semgid, sh->requestReceived[sh->fSt.assignedTable[n]]) == -1) {
30        perror("error on the up operation for semaphore access (WT-REQUESTRECEIVED)");
31        exit(EXIT_FAILURE);
32    }
33 }
```

Na terceira função, o estado da função é atualizado para `TAKE_TO_TABLE`, e o empregado de mesa entrega a comida ao grupo.

```
1 static void takeFoodToTable(int n) {
2     if (semDown(semgid, sh->mutex) == -1) { /* enter critical region */
3         perror("error on the down operation for semaphore access (WT)");
4         exit(EXIT_FAILURE);
5     }
6
7     // TODO insert your code here
8     sh->fSt.waiterStat = TAKE_TO_TABLE;
9     saveState(nFic, &sh->fSt);
10
11    if (semUp(semgid, sh->mutex) == -1) { /* exit critical region */
12        perror("error on the up operation for semaphore access (WT)");
13        exit(EXIT_FAILURE);
14    }
15
16    if (semUp(semgid, sh->foodArrived[sh->fSt.assignedTable[n]]) == -1) {
17        perror("error on the up operation for semaphore access (WT-FOODARRIVED)");
18        exit(EXIT_FAILURE);
19    }
20 }
```

1.4 – Chef

Por um lado, o chef, na função `waitForOrder`, espera que cheguem pedidos e, quando há, atualiza o seu estado para `COOK`, confirma a receção do pedido para que o empregado de mesa tenha esse conhecimento e o passe ao grupo.

```

1 static void waitForOrder() {
2     // TODO insert your code here
3     if (semDown(semgid, WAITORDER) == -1) {
4         perror("error on the down operation for semaphore access(CH-WAITORDER)");
5         exit(EXIT_FAILURE);
6     }
7
8     if (semDown(semgid, sh->mutex) == -1) { /* enter critical region */
9         perror("error on the down operation for semaphore access (CH)");
10        exit(EXIT_FAILURE);
11    }
12
13    // TODO insert your code here
14    sh->fSt.st.chefStat = COOK;
15    lastGroup = sh->fSt.foodGroup;
16    saveState(nFic, &sh->fSt);
17
18    if (semUp(semgid, sh->mutex) == -1) { /* exit critical region */
19        perror("error on the up operation for semaphore access (CH)");
20        exit(EXIT_FAILURE);
21    }
22
23    // TODO insert your code here
24    if (semUp(semgid, ORDERRECEIVED) == -1) {
25        perror("error on the up operation for semaphore access (CH-ORDERRECEIVED)");
26        exit(EXIT_FAILURE);
27    }
28 }

```

Por outro lado, na função `processOrder`, o chef processa o pedido e aguarda que o empregado de mesa esteja disponível, atualiza o seu estado para `WAIT_FOR_ORDER` e indica que a comida está pronta.

```

1 static void processOrder() {
2     usleep((unsigned int)floor((MAXCOOK * random()) / RAND_MAX + 100.0));
3
4     // TODO insert your code here
5     if (semDown(semgid, WAITERREQUESTPOSSIBLE) == -1) {
6         perror("error on the down operation for semaphore access (CH-WAITERREQUESTPOSSIBLE)");
7         exit(EXIT_FAILURE);
8     }
9
10    if (semDown(semgid, sh->mutex) == -1) { /* enter critical region */
11        perror("error on the down operation for semaphore access (CH)");
12        exit(EXIT_FAILURE);
13    }
14
15    // TODO insert your code here
16    sh->fSt.st.chefStat = WAIT_FOR_ORDER;
17    sh->fSt.waiterRequest.reqType = FOODREADY;
18    sh->fSt.waiterRequest.reqGroup = lastGroup;
19    sh->fSt.foodOrder = 0;
20    saveState(nFic, &sh->fSt);
21
22    if (semUp(semgid, sh->mutex) == -1) { /* exit critical region */
23        perror("error on the up operation for semaphore access (CH)");
24        exit(EXIT_FAILURE);
25    }
26
27    // TODO insert your code here
28    if (semUp(semgid, WAITERREQUEST) == -1) {
29        perror("error on the up operation for semaphore access (CH-WAITERREQUEST)");
30        exit(EXIT_FAILURE);
31    }
32 }

```

2 – Testagem

De modo a confirmar se os resultados que obtivemos estavam de acordo com o espectável, verificámos se existia algum deadlock, pelo que constatámos que eram inexistentes e analisámos vários resultados de várias execuções, como mostra o seguinte exemplo:

Restaurant - Description of the internal state																		
CH		WT		RC		G00	G01	G02	G03	G04	gWT	T00	T01	T02	T03	T04		
0	WAIT_FOR_ORDER	0	*	0	*	1	1	1	1	1	0		
0		0		1		1	1	1	0				
0		0		1		1	1	1	0				
0		0		1		1	1	1	0				
0		0		1	ASSIGNTABLE	1	1	2	1	1	0		
0		0		1		1	2	1	1	0	.	.	0	.	.			
0		0		1		1	2	1	1	0	.	.	0	.	.			
0		0		1		1	3	1	1	0	.	.	0	.	.			
0		0		1		1	3	1	1	0	.	.	0	.	.			
0		0		1		1	4	1	1	0	.	.	0	.	.			
1	COOK	1	INFORM_CHEF	0	*	1	1	4	1	1	0	.	.	0	.	.		
1		0		1		1	4	1	1	0	.	.	0	.	.			
1		0		1		2	4	1	1	0	.	.	0	.	.			
1		0		1		2	4	1	1	0	.	.	0	.	.			
0		0		1		2	4	1	1	0	.	1	0	.	.			
0		0		1		2	4	1	1	0	.	1	0	.	.			
0		WAIT_FOR_ORDER		2	TAKE_TO_TABLE	1	ASSIGNTABLE	1	2	4	1	1	0	.	1	0	.	.
0				2		1		3	4	1	1	0	.	1	0	.	.	
0				2		0		3	4	1	1	0	.	1	0	.	.	
0				0		1		3	5	1	1	0	.	1	0	.	.	
0	0		1	3		5		1	1	0	.	1	0	.	.			
0	1		0	4		5		1	1	0	.	1	0	.	.			
1	COOK		1	INFORM_CHEF		0	*	1	4	5	1	1	0	.	1	0	.	.
1			0			1		4	5	1	1	0	.	1	0	.	.	
0			0			1		4	5	1	1	0	.	1	0	.	.	
0			1			0		4	5	1	1	0	.	1	0	.	.	
0		1	0		4	5		1	1	0	.	1	0	.	.			
0		0	1		4	5		1	1	0	.	1	0	.	.			
0		WAIT_FOR_ORDER	2		TAKE_TO_TABLE	0	*	1	4	5	1	1	0	.	1	0	.	.
0			0			1		4	5	1	1	0	.	1	0	.	.	
0			0			1		5	5	1	1	0	.	1	0	.	.	
0			0			1		5	5	2	1	1	.	1	0	.	.	
0	0		1	5		5		2	1	1	.	1	0	.	.			
0	0		1	5		5		2	2	1	.	1	0	.	.			
0	0		1	5		5		2	2	2	.	1	0	.	.			
0	0		1	5		5		2	2	2	.	1	0	.	.			
0	WAIT_FOR_ORDER	0	*	0	ASSIGNTABLE	2		5	5	2	2	2	.	1	0	.	.	
0		0		2		5		5	2	2	2	.	1	0	.	.		
0		0		2		5	5	2	2	3	.	1	0	.	.			
0		0		2		5	5	2	2	3	.	1	0	.	.			
0		0		2		5	6	2	2	3	.	1	0	.	.			
0		0		2		5	6	2	2	2	0	1	.	.	.			
0		0		2		5	6	2	2	2	0	1	.	.	.			
0		0		2		5	6	2	2	2	0	1	.	.	.			

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

* - WAIT_FOR_REQUEST

gWt = nº de grupos à espera

Legenda para Group:

1-GOTOREST
 2-ATRECEPTION
 3-FOOD_REQUEST
 4-WAIT_FOR_FOOD
 5-EAT
 6-CHECKOUT
 7-LEAVING

Legenda:

0 e 1 são os ids das mesas

A fim de verificar se existia algum deadlock, com o auxílio do script `run.sh`, corremos o programa múltiplas vezes, verificando que nunca ocorreu nenhum deadlock.

Na figura anterior conseguimos ver como interagem as várias entidades num restaurante com duas mesas e cinco grupos.

CH, WT e RC são, respetivamente, o Chef, o Waiter (empregado de mesa) e Rececionista. G00-G04 representa cada um dos grupos que foram ao restaurante. gWT corresponde ao nº de grupos à espera por uma mesa para jantarem.

Para melhor aferirmos a fiabilidade dos resultados alterámos múltiplas vezes o número de grupos que foram ao restaurante. As análises realizadas foram feitas afim de verificar linha a linha se as mudanças de estado faziam sentido.

Comparámos algumas das nossas execuções com o resultado da execução pré-compilada do professor e, como era de esperar, observámos que eram bastante semelhantes.

Conclusão

Os conhecimentos para a realização deste trabalho foram adquiridos essencialmente nas aulas práticas e teóricas, não tendo sido necessário qualquer pesquisa da nossa parte.

Um dos maiores desafios que tivemos foi perceber a interação entre as entidades e quais os semáforos a usar aquando dessa interação, isto é, quais os semáforos que deveriam estar ativos e/ou bloqueados, e como poderíamos confirmar se o que fizemos estaria em conformidade com o proposto.

Os programas foram implementados com sucesso, não havendo nenhum deadlock e, tendo passo em todos os testes a que foram propostos.

Este trabalho permitiu-nos desenvolver habilidades de organização, gestão de tempo e perseverança. Enfrentámos desafios ao longo do caminho, mas com dedicação e determinação conseguimos superá-los.