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

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	Termina de processar um request pendente
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	Quer processar um pedido/ Pretende encomendar o pedido	waiter	Termina de processar um request pendente
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	Espera pela confirmação do pagamento	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).

```c
static void checkInAtReception(int id) {
  // TODO insert your code here
  if (semDown(semgid, RECEPTIONISTREQUESTPOSSIBLE) == -1) {                    (1)
    perror("error on the down operation for semaphore access (GR-RECEPTIONISTREQUESTPOSSIBLE)");
    exit(EXIT_FAILURE);
  }

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

  // TODO insert your code here
  sh->fSt.st.groupStat[id] = ATRECEPTION;                                      (2)
  sh->fSt.receptionistRequest.reqGroup = id;
  sh->fSt.receptionistRequest.reqType = TABLEREQ;                              (3)
  saveState(nFic, &sh->fSt);

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

  // TODO insert your code here

  if (semUp(semgid, RECEPTIONISTREQ) == -1) {                                  (4)
    perror("error on the up operation for semaphore access (GR-RECEPTIONISTREQ)");
    exit(EXIT_FAILURE);
  }

  if (semDown(semgid, sh->waitForTable[id]) == -1) {                           (5)
    perror("error on the down operation for semaphore access (GR-WAITFORTABLE)");
    exit(EXIT_FAILURE);
  }
}
```

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).

```c
static void orderFood(int id) {
  // TODO insert your code here
  if (semDown(semgid, WAITERREQUESTPOSSIBLE) == -1) {                          (1)
    perror("error on the down operation for semaphore access (GR-WAITERREQUESTPOSSIBLE)");
    exit(EXIT_FAILURE);
  }

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

  // TODO insert your code here
  sh->fSt.st.groupStat[id] = FOOD_REQUEST;                                     (2)
  sh->fSt.waiterRequest.reqGroup = id;
  sh->fSt.waiterRequest.reqType = FOODREQ;                                     (3)
  saveState(nFic, &sh->fSt);

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

  // TODO insert your code here
  if (semUp(semgid, WAITERREQUEST) == -1) {                                    (4)
    perror("error on the up operation for semaphore access (GR-WAITERREQUEST)");
    exit(EXIT_FAILURE);
  }

  if (semDown(semgid, sh->requestReceived[sh->fSt.assignedTable[id]]) == -1) { (5)
    perror("error on the down operation for semaphore access (GR-REQUESTRECEIVED)");
    exit(EXIT_FAILURE);
  }
}
```

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;                                    (1)
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) {     (2)
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;                                              (3)
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) {                  (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                                               (2)
12   sh->fSt.st.groupStat[id] = CHECKOUT;
13   sh->fSt.receptionistRequest.reqGroup = id;
14   sh->fSt.receptionistRequest.reqType = BILLREQ;                             (3)
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) {                                (4)
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) {     (5)
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;                                        (6)
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.

```c
static request waitForGroup() {
  request ret;

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

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

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

  // TODO insert your code here
  if (semDown(semgid, RECEPTIONISTREQ) == -1) {
    perror("error on the down operation for semaphore access (RC-RECEPTIONISTREQ)");
    exit(EXIT_FAILURE);
  }

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

  // TODO insert your code here
  ret = sh->fSt.receptionistRequest;

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

  // TODO insert your code here
  if (semUp(semgid, RECEPTIONISTREQUESTPOSSIBLE) == -1) {
    perror("error on the up operation for semaphore access (RC-RECEPTIONISTREQUESTPOSSIBLE)");
    exit(EXIT_FAILURE);
  }

  return ret;
}
```

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.

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

  // TODO insert your code here
  sh->fSt.st.receptionistStat = ASSIGNTABLE;

  int table = decideTableOrWait(n);

  if (table == -1) {
    sh->fSt.groupsWaiting++;
    groupRecord[n] = WAIT;
  } else {
    sh->fSt.assignedTable[n] = table;
    groupRecord[n] = ATTABLE;
    if (semUp(semgid, sh->waitForTable[n]) == -1) {
      perror("error on the up operation for semaphore access (RC-WAITFORTABLE)");
      exit(EXIT_FAILURE);
    }
  }

  saveState(nFic, &sh->fSt);

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

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.

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

  // TODO insert your code here
  sh->fSt.st.receptionistStat = RECVPAY;
  groupRecord[n] = DONE;
  if (semUp(semgid, sh->tableDone[sh->fSt.assignedTable[n]]) == -1) {
    perror("error on the up operation for semaphore access (RC-TABLEDONE)");
    exit(EXIT_FAILURE);
  }

  int next_group = decideNextGroup();

  if (next_group != -1) {
    groupRecord[next_group] = ATTABLE;
    sh->fSt.assignedTable[next_group] = sh->fSt.assignedTable[n];
    sh->fSt.assignedTable[n] = -1;
    sh->fSt.groupsWaiting--;
  }

  saveState(nFic, &sh->fSt);

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

  // TODO insert your code here
  if (next_group != -1) {
    if (semUp(semgid, sh->waitForTable[next_group]) == -1) {
      perror("error on the up operation for semaphore access (RC-WAITFORTABLE)");
      exit(EXIT_FAILURE);
    }
  }
```

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.

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

  // TODO insert your code here
  // Updates the waiter stat to wait for a request
  sh->fSt.st.waiterStat = WAIT_FOR_REQUEST;
  saveState(nFic, &sh->fSt);

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

  // TODO insert your code here
  // Waits until the waiter receives a request
  if (semDown(semgid, WAITERREQUEST) == -1) {
    perror("error on the down operation for semaphore access (WT-WAITERREQUEST)");
    exit(EXIT_FAILURE);
  }

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

  // TODO insert your code here
  req = sh->fSt.waiterRequest;

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

  // TODO insert your code here
  if (semUp(semgid, WAITERREQUESTPOSSIBLE) == -1) {
    perror("error on the up operation for semaphore access (WT-WAITERREQUESTPOSSIBLE)");
    exit(EXIT_FAILURE);
  }

  return req;
}
```

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.

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

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

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

  // TODO insert your code here
  // Wakes up the chef
  if (semUp(semgid, WAITORDER) == -1) {
    perror("error on the up operation for semaphore access (WT-WAITORDER)");
    exit(EXIT_FAILURE);
  }
  if (semDown(semgid, ORDERRECEIVED) == -1) {
    perror("error on the down operation for semaphore access (WT-ORDERRECEIVED)");
    exit(EXIT_FAILURE);
  }
  if (semUp(semgid, sh->requestReceived[sh->fSt.assignedTable[n]]) == -1) {
    perror("error on the up operation for semaphore access (WT-REQUESTRECEIVED)");
    exit(EXIT_FAILURE);
  }
}
```

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

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

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

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

  if (semUp(semgid, sh->foodArrived[sh->fSt.assignedTable[n]]) == -1) {
    perror("error on the up operation for semaphore access (WT-FOODARRIVED)");
    exit(EXIT_FAILURE);
  }
}
```

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.

```c
static void waitForOrder() {
  // TODO insert your code here
  if (semDown(semgid, WAITORDER) == -1) {
    perror("error on the down operation for semaphore access(CH-WAITORDER)");
    exit(EXIT_FAILURE);
  }

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

  // TODO insert your code here
  sh->fSt.st.chefStat = COOK;
  lastGroup = sh->fSt.foodGroup;
  saveState(nFic, &sh->fSt);

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

  // TODO insert your code here
  if (semUp(semgid, ORDERRECEIVED) == -1) {
    perror("error on the up operation for semaphore access (CH-ORDERRECEIVED)");
    exit(EXIT_FAILURE);
  }
}
```

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.

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

  // TODO insert your code here
  if (semDown(semgid, WAITERREQUESTPOSSIBLE) == -1) {
    perror("error on the down operation for semaphore access (CH-WAITERREQUESTPOSSIBLE)");
    exit(EXIT_FAILURE);
  }

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

  // TODO insert your code here
  sh->fSt.st.chefStat = WAIT_FOR_ORDER;
  sh->fSt.waiterRequest.reqType = FOODREADY;
  sh->fSt.waiterRequest.reqGroup = lastGroup;
  sh->fSt.foodOrder = 0;
  saveState(nFic, &sh->fSt);

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

  // TODO insert your code here
  if (semUp(semgid, WAITERREQUEST) == -1) {
    perror("error on the up operation for semaphore access (CH-WAITERREQUEST)");
    exit(EXIT_FAILURE);
  }
}
```

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		0		1	1	1	1	1	0	.	.	.	.	.
0		0		0		1	1	1	1	1	0	.	.	.	.	.
0		0		0		1	1	2	1	1	0	.	.	.	.	.
0		0		1	ASSIGNTABLE	1	1	2	1	1	0	.	.	0	.	.
0		0		0	*	1	1	2	1	1	0	.	.	0	.	.
0		0		0		1	1	3	1	1	0	.	.	0	.	.
0		1	INFORM_CHEF	0		1	1	3	1	1	0	.	.	0	.	.
0		1		0		1	1	4	1	1	0	.	.	0	.	.
1	COOK	1		0		1	1	4	1	1	0	.	.	0	.	.
1		0	*	0		1	1	4	1	1	0	.	.	0	.	.
1		0		0		1	2	4	1	1	0	.	.	0	.	.
0		0		0		1	2	4	1	1	0	.	.	0	.	.
0	WAIT_FOR_ORDER	0		1		1	2	4	1	1	0	.	1	0	.	.
0		2	TAKE_TO_TABLE	1	ASSIGNTABLE	1	2	4	1	1	0	.	1	0	.	.
0		2		1		1	3	4	1	1	0	.	1	0	.	.
0		2		0		1	3	4	1	1	0	.	1	0	.	.
0		0	*	0		1	3	4	1	1	0	.	1	0	.	.
0		0		0		1	3	5	1	1	0	.	1	0	.	.
0		1	INFORM_CHEF	0		1	3	5	1	1	0	.	1	0	.	.
0		1		0		1	4	5	1	1	0	.	1	0	.	.
1	COOK	1		0		1	4	5	1	1	0	.	1	0	.	.
1		0	*	0	*	1	4	5	1	1	0	.	1	0	.	.
0		0		0		1	4	5	1	1	0	.	1	0	.	.
0		2	TAKE_TO_TABLE	0		1	4	5	1	1	0	.	1	0	.	.
0		0		0		1	4	5	1	1	0	.	1	0	.	.
0		0		0		1	5	5	1	1	0	.	1	0	.	.
0		0		0		1	5	5	2	1	0	.	1	0	.	.
0		0		1	ASSIGNTABLE	1	5	5	2	1	1	.	1	0	.	.
0		0		0	*	1	5	5	2	1	1	.	1	0	.	.
0		0		0		1	5	5	2	2	1	.	1	0	.	.
0	WAIT_FOR_ORDER	0		1	ASSIGNTABLE	1	5	5	2	2	2	.	1	0	.	.
0		0		0	*	1	5	5	2	2	2	.	1	0	.	.
0		0	*	0		2	5	5	2	2	2	.	1	0	.	.
0		0		1	ASSIGNTABLE	2	5	5	2	2	3	.	1	0	.	.
0		0		0	*	2	5	5	2	2	3	.	1	0	.	.
0		0		0		2	5	6	2	2	3	.	1	0	.	.
0		0		2	RECVPAY	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		0		3	5	7	2	2	2	0	1	.	.	.
0		1	INFORM_CHEF	0		3	5	7	2	2	2	0	1	.	.	.
0		1		0		4	5	7	2	2	2	0	1	.	.	.
1	COOK	1		0		4	5	7	2	2	2	0	1	.	.	.
1		0	*	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	0		4	5	7	2	2	2	0	1	.	.	.
0	WAIT_FOR_ORDER	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	*	2	RECVPAY	6	5	7	2	2	1	.	1	.	0	.
0		0		2		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	0		7	5	7	3	2	1	.	1	.	0	.
0		1		0		7	5	7	4	2	1	.	1	.	0	.
1	COOK	1		0		7	5	7	4	2	1	.	1	.	0	.
1		0	*	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	0		7	5	7	4	2	1	.	1	.	0	.
0	WAIT_FOR_ORDER	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	*	2	RECVPAY	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	0		7	5	7	7	3	0	.	1	.	.	0
0		1		0		7	5	7	7	4	0	.	1	.	.	0
1	COOK	1		0	*	7	5	7	7	4	0	.	1	.	.	0
1		0	*	0		7	5	7	7	4	0	.	1	.	.	0
0		0		0		7	5	7	7	4	0	.	1	.	.	0
0		2	TAKE_TO_TABLE	0		7	5	7	7	4	0	.	1	.	.	0
0	WAIT_FOR_ORDER	2		0		7	5	7	7	5	0	.	1	.	.	0
0		2		0		7	5	7	7	6	0	.	1	.	.	0
0		2		2	RECVPAY	7	5	7	7	6	0	.	1	.	.	.
0		2		0		7	5	7	7	6	0	.	1	.	.	.
0		2		0	*	7	5	7	7	7	0	.	1	.	.	.
0		2		0		7	6	7	7	7	0	.	1	.	.	.
0		2		2	RECVPAY	7	6	7	7	7	0	.	.	.	.	.
0		2		2		7	7	7	7	7	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.