

Sistemas Operativos

Ano letivo

universidade de aveiro  **deti** departamento de electrónica,
telecomunicações e informática

2018/2019

Docente: Nuno Lau

Relatório do Segundo Trabalho de SO

Restaurante (Semáforos)

João Dias, nº89236

Tomás Batista, nº89296

Índice

Introdução	4
Chef.....	5
Waiter.....	8
Grupo.....	12
Rececionista.....	18
Ferramentas de Suporte.....	24

Introdução

Neste trabalho da unidade curricular de Sistemas Operativos, procurámos construir uma aplicação em C que simule o processo que ocorre num restaurante. Neste estão envolvidas 4 entidades: os grupos (de clientes), o rececionista, o waiter e o chef.

Estas entidades terão de comunicar entre si de modo a desempenhar as funções necessárias seguindo as normas estipuladas no enunciado.

Cada grupo começa por dirigir-se ao rececionista, este vai indicar ao grupo se há uma mesa disponível ou se tem de esperar, após obter mesa, o grupo pede a comida ao waiter, este leva o pedido ao chef e este prepara a comida. Quando a comida está pronta, o waiter traz a comida para a mesa. Ao fim de todos comerem, o grupo contacta novamente o rececionista para efetuar o pagamento. Por fim, o grupo abandona o restaurante-

Chef

O *chef* visto que é a entidade com menos estados e interações, foi a entidade pela qual começámos a trabalhar. O seu papel no programa consiste em esperar que o *waiter* traga os pedidos, cozinhar e entregar a comida pronta ao *waiter*.

```
/* Chef state constants */

/** \brief chef waits for food order */
#define WAIT_FOR_ORDER 0
/** \brief chef is cooking */
#define COOK 1
/** \brief chef is resting */
#define REST 2
```

Estados do Chef

Comentado [JD1]:

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

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

    // Here the chef status must be updated
    sh->fSt.foodOrder = 0;
    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 (PT)");
        exit(EXIT_FAILURE);
    }
}
```

```

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

```

Função waitForOrder()

Na função waitForOrder() começa-se por dar down() no semáforo waitOrder, indicando, desta maneira, que está à espera de um pedido.

Caso não tenha sido efetuado nenhum pedido, o valor de foodOrder vai ser igual a 0 e a função termina, caso isto não ocorra, o valor da flag foodOrder é alterado para 1, este vai manter-se assim até um novo pedido chegar. O estado do chef é alterado para "COOK", sendo que o pedido já foi recebido.

```

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

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

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

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

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

```

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

Função processOrder()

Na função processOrder() inicia com uma função que simula o tempo de preparação do pedido, após o pedido estar preparado, o *waiter* é avisado que o pedido já foi preparado e o *chef* muda de estado para WAIT_FOR_ORDER.

Waiter

O waiter foi a segunda entidade em que trabalhámos, visto que interage com o chef, que tinha sido previamente feito, e com o grupo.

O seu papel no programa é levar os pedidos dos grupos para o chef e a comida do chef para os grupos.

```
/* Waiter state constants */

/** \brief waiter waits for food request */
#define WAIT_FOR_REQUEST 0
/** \brief waiter takes food request to chef */
#define INFORM_CHEF 1
/** \brief waiter takes food to table */
#define TAKE_TO_TABLE 2
```

Estados do waiter

```
static request waitForClientOrChef()
{
    request req;

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

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

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

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



```

}

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

if (sh->fSt.waiterRequest.reqType == FOODREQ)
{
    req.reqType = FOODREQ;
}
if (sh->fSt.waiterRequest.reqType == FOODREADY)
{
    req.reqType = FOODREADY;
}

req.reqGroup = sh->fSt.waiterRequest.reqGroup;

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

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

return req;
}

```

Função waitForClientOrChef()

Na função waitForClientOrChef(), o waiter espera por um request do chef com um pedido pronto, ou do grupo com um novo pedido. O valor de retorno vai determinar que função executar posteriormente.

A função inicia com uma alteração do estado do waiter para WAIT_FOR_REQUEST e é feito down() ao semáforo waiterRequest.

Posteriormente é feita a verificação do tipo de request que foi feito.

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

    sh->fSt.foodOrder = 1;
    sh->fSt.st.waiterStat = INFORM_CHEF;
    sh->fSt.foodGroup = n;
    saveState(nFic, &sh->fSt);

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

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

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

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

Função informChef(int id)

Na função `informChef()` o objetivo é entregar ao chef o pedido, a função começa por alterar o valor de `foodOrder`, o estado do waiter é também alterado para `INFORM_CHEF` e o valor de `foodGroup` é definido.

Posteriormente é desbloqueado o semáforo `waitOrder` e `orderReceiver` pois o chef já não se encontra à espera de um pedido.

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

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

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

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

Função `takeFoodToTable(int id)`

A função `takeFoodToTable(int n)` tem como objetivo o waiter levar a comida para a mesa. O estado do waiter é alterado para `TAKE_TO_TABLE` e os semáforos `foodArrived` e `assignedTable`, associados a cada elemento do grupo, são desbloqueados.

Grupo

O grupo foi a terceira entidade que realizámos, este interage com o rececionista e com o waiter.

```
/* Client state constants */

/** \brief group initial state */
#define GTOREST 1
/** \brief client is waiting at reception or waiting for table */
#define ATRECEPTION 2
/** \brief client is requesting food to waiter */
#define FOOD_REQUEST 3
/** \brief client is waiting for food */
#define WAIT_FOR_FOOD 4
/** \brief client is eating */
#define EAT 5
/** \brief client is checking out */
#define CHECKOUT 6
/** \brief client is leaving */
#define LEAVING 7
```

Estados do grupo

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

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

    sh->fSt.st.groupStat[id] = ATRECEPTION;
    saveState(nFic, &sh->fSt);
}
```

```

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);
}
/* exit critical region */

if (semUp(semgid, sh->receptionistReq) == -1)
{
    perror("error on the up operation for semaphore access (CT)");
    exit(EXIT_FAILURE);
}
if (semDown(semgid, sh->waitForTable[id]) == -1)
{
    perror("error on the up operation for semaphore access (CT)");
    exit(EXIT_FAILURE);
}
}

```

Função checkInAtReception(int id)

A função checkInAtReception (int id) tem como objetivo obter uma das duas respostas, se o grupo vai ter de esperar por uma mesa ou se vai de imediato ficar com uma mesa atribuída.

É atribuído o estado ATRECEPTION a todos os elementos do grupo e é enviado para o rececionista um TABLEREQ e o id do grupo.

```

static void orderFood(int id)
{
    if (semDown(semgid, sh->waiterRequestPossible) == -1)
    {
        perror("error on the down operation for semaphore access (CT)");
        exit(EXIT_FAILURE);
    }

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

```

```

}

sh->fSt.st.groupStat[id] = FOOD_REQUEST;
saveState(nFic, &sh->fSt);
sh->fSt.waiterRequest.reqType = FOODREQ;
sh->fSt.waiterRequest.reqGroup = id;
int RC_id = sh->fSt.assignedTable[id];

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

if (semUp(semgid, sh->waiterRequest) == -1)
{
    perror("error on the up operation for semaphore access (CT)");
    exit(EXIT_FAILURE);
}
if (semDown(semgid, sh->requestReceived[RC_id]) == -1)
{
    perror("error on the down operation for semaphore access (CT)");
    exit(EXIT_FAILURE);
}
}

```

Função orderFood(int id)

Na função orderFood(id) é quando é feito o pedido da comida, o estado de cada elemento do grupo muda para FOOD_REQUEST, é enviado também para o waiter um FOODREQ e o id da mesa.

```

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

    sh->fSt.st.groupStat[id] = WAIT_FOR_FOOD;
    saveState(nFic, &sh->fSt);
    int RC_id = sh->fSt.assignedTable[id];
}

```

```

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

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

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

sh->fSt.st.groupStat[id] = EAT;
saveState(nFic, &sh->fSt);

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

```

Função waitFood(int id)

A função waitFood(int id) é efetuada por todos os elementos do grupo, o estado deles é inicialmente alterado para WAIT_FOR_FOOD, foodArrived é bloqueado e quando a comida chega, o estado de cada elemento do grupo muda para EAT.

```

static void checkOutAtReception(int id)
{
    if (semDown(semgid, sh->receptionistRequestPossible) == -1)
    {
        perror("error on the down operation for semaphore access (CT)");
        exit(EXIT_FAILURE);
    }

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

    sh->fSt.st.groupStat[id] = CHECKOUT;
    saveState(nFic, &sh->fSt);
    sh->fSt.receptionistRequest.reqType = BILLREQ;
    sh->fSt.receptionistRequest.reqGroup = id;
    int RC_id = sh->fSt.assignedTable[id];

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

    if (semUp(semgid, sh->receptionistReq) == -1)
    {
        perror("error on the down operation for semaphore access (CT)");
        exit(EXIT_FAILURE);
    }
    if (semDown(semgid, sh->tableDone[RC_id]) == -1)
    {
        perror("error on the down operation for semaphore access (CT)");
        exit(EXIT_FAILURE);
    }

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

    sh->fSt.st.groupStat[id] = LEAVING;
    saveState(nFic, &sh->fSt);

    if (semUp(semgid, sh->mutex) == -1)

```



```
{
    perror("error on the down operation for semaphore access (CT)");
    exit(EXIT_FAILURE);
}
/* exit critical region */
}
```

Função checkOutAtReception(int id)

A função `checkOutAtReception(int id)` é corrida por cada elemento do grupo quando acabem de comer, o estado de cada elemento muda para CHECKOUT e é enviado ao receptionista um BILLREQ e o id. Finalmente o semáforo `tableDone` é desbloqueado.

Rececionista

O rececionista foi a última entidade que realizámos, esta só interage com os grupos, atribui-lhes uma mesa caso disponível ou dá-lhes a informação de que têm de esperar. O rececionista é também o encarregue de receber o pagamento no final da refeição.

```
/* Receptionist state constants */

/** \brief waiter waits for food request */
#define ASSIGNTABLE 1
/** \brief waiter receives payment */
#define RECVPAY 2
```

Estados do rececionista

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

    sh->fSt.st.receptionistStat = WAIT_FOR_REQUEST;
    saveState(nFic, &sh->fSt);

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

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

    /* enter critical region */
    if (semDown(semgid, sh->mutex) == -1)
```

```

{
    perror("error on the up operation for semaphore access (WT)");
    exit(EXIT_FAILURE);
}

ret.reqGroup = sh->fSt.receptionistRequest.reqGroup;

if (sh->fSt.receptionistRequest.reqType == TABLEREQ)
{
    ret.reqType = TABLEREQ;
}
if (sh->fSt.receptionistRequest.reqType == BILLREQ)
{
    ret.reqType = BILLREQ;
}

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

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

return ret;
}

```

Função waitForGroup()

Na função waitForGroup () o rececionista espera por um pedido, TABLEREQ ou BILLREQ, no início da função o estado do rececionista é alterado para WAIT_FOR_REQUEST até um pedido ser recebido, a função que executa após esta depende do tipo de pedido.

```

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

    if (decideTableOrWait(n) > -1)
    {
        sh->fSt.st.receptionistStat = ASSIGNTABLE;
        sh->fSt.assignedTable[n] = decideTableOrWait(n);
        groupRecord[n] = ATTABLE;
        if (semUp(semgid, sh->waitForTable[n]) == -1)
        {
            perror("error on the up operation for semaphore access (WT)");
            exit(EXIT_FAILURE);
        }
    }
    else
    {
        sh->fSt.groupsWaiting++;
        groupRecord[n] = WAIT;
    }
    saveState(nFic, &sh->fSt);

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

```

Função provideTableOrWaitingRoom(int n)

Na função provideTableOrWaitingRoom(int n) é onde é decidido se o grupo fica com uma mesa atribuída ou vai para a sala de espera, caso haja uma mesa livre o estado do rececionista muda para ASSIGNTABLE e o grupo passa para a mesa atribuída. Caso contrário, o grupo passa para WAIT, incrementando por um o número de grupos à espera.

```

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

    sh->fSt.st.receptionistStat = RECVPAY;
    groupRecord[n] = DONE;
    saveState(nFic, &sh->fSt);

    int assignedTable = sh->fSt.assignedTable[n];

    if (sh->fSt.groupsWaiting > 0)
    {
        int nextGroup = decideNextGroup();
        sh->fSt.assignedTable[nextGroup] = sh->fSt.assignedTable[n];
        groupRecord[nextGroup] = ATTABLE;
        sh->fSt.st.receptionistStat = WAIT_FOR_REQUEST;
        sh->fSt.groupsWaiting--;
        if (semUp(semgid, sh->waitForTable[nextGroup]) == -1)
        {
            perror("error on the up operation for semaphore access (WT)");
            exit(EXIT_FAILURE);
        }
    }
    sh->fSt.assignedTable[n] = -1;

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

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

```

Função receivePayment(int n)

Os grupos efetuam o pagamento e passam para o estado DONE, o rececionista fica no estado RECVPAY e o estado é guardado. Caso

estejam grupos à espera o rececionista irá dar-lhes uma mesa, o grupo passa para o estado ATTABLE e o rececionista passa para o estado WAIT_FOR_REQUEST.

Conclusão

A realização deste projeto permitiu, de uma melhor maneira, ver a importância e o poder de semáforos na realização de tarefas que envolvem várias entidades.

Dado o exposto e após a sua execução, todas as interações e resultados pretendidos entre entidades foram conseguidos, logo, podemos dizer que a realização deste projeto foi bem-sucedida.

Restaurant - Description of the internal state										
CH	WT	RC	G00	G01	G02	gWT	T00	T01	T02	
0	0	0	1	1	1	0	.	.	.	
0	0	0	1	1	1	0	.	.	.	
0	0	0	1	1	2	0	.	.	.	
0	0	0	1	1	2	0	.	.	.	
0	0	1	1	1	2	0	.	.	0	
0	0	0	1	1	2	0	.	.	0	
0	0	0	1	1	3	0	.	.	0	
0	1	0	1	1	3	0	.	.	0	
0	1	0	1	1	4	0	.	.	0	
1	1	0	1	1	4	0	.	.	0	
1	0	0	1	1	4	0	.	.	0	
0	0	0	1	1	4	0	.	.	0	
0	0	0	2	1	4	0	.	.	0	
0	2	0	2	1	4	0	.	.	0	
0	0	0	2	1	4	0	.	.	0	
0	0	1	2	1	4	0	1	.	0	
0	0	1	3	1	4	0	1	.	0	
0	0	0	3	1	4	0	1	.	0	
0	0	0	3	1	5	0	1	.	0	
0	1	0	3	1	5	0	1	.	0	
1	1	0	3	1	5	0	1	.	0	
1	0	0	3	1	5	0	1	.	0	
1	0	0	4	1	5	0	1	.	0	
0	0	0	4	1	5	0	1	.	0	
0	2	0	4	1	5	0	1	.	0	
0	0	0	4	1	5	0	1	.	0	
0	0	0	5	1	5	0	1	.	0	
0	0	0	6	1	5	0	1	.	0	
0	0	2	6	1	5	0	1	.	0	
0	0	0	6	1	5	0	.	.	0	
0	0	0	7	1	5	0	.	.	0	
0	0	0	7	2	5	0	.	.	0	
0	0	1	7	2	5	0	.	1	0	
0	0	0	7	2	5	0	.	1	0	
0	0	0	7	3	5	0	.	1	0	
0	1	0	7	3	5	0	.	1	0	
1	1	0	7	3	5	0	.	1	0	
1	0	0	7	3	5	0	.	1	0	
1	0	0	7	4	5	0	.	1	0	
0	0	0	7	4	5	0	.	1	0	
0	2	0	7	4	5	0	.	1	0	
0	2	0	7	5	5	0	.	1	0	
0	2	0	7	5	6	0	.	1	0	
0	2	2	7	5	6	0	.	1	0	
0	2	2	7	6	6	0	.	1	.	
0	2	0	7	6	6	0	.	1	.	
0	2	0	7	6	7	0	.	1	.	
0	2	2	7	6	7	0	.	1	.	
0	2	2	7	7	7	0	.	.	.	

Ferramentas de Suporte

- [GitHub](#)
- [Code UA](#)