universidade de aveiro  theoria poiesis praxis

# Trabalho 2

## Simulação de Ponte Aérea

**Universidade de Aveiro**

Licenciatura em Engenharia Informática

Sistemas Operativos

1 fevereiro 2022

Docentes: Guilherme Campos e Nuno Lau

Pedro Jorge, Nº Mecanográfico: 98418

# Índice

**Introdução** 3

**O problema**

Descrição e Considerações 4/5
Variáveis gerais e Estados 6
Estruturas de Dados 9
Semáforos Pré-Definidos 11
Tabela de Semáforos 12

**Implementação**

Piloto 13
Hospedeira 17
Passageiro 23

**Resultados Obtidos** 27

**Conclusão** 30

**Bibliografia** 31

# Introdução

Como segundo e último projeto da cadeira de Sistemas Operativos foi proposto desenvolver uma aplicação, na linguagem de programação C, que simule uma ponte aérea, envolvendo 3 entidades: passageiros, hospedeira e piloto. Todos estes processos são processos independentes, sendo a sua sincronização realizada através de semáforos e memória partilhada. Todos os processos são criados a partir do programa **probSemSharedMemAirLift** e estão em execução a partir dessa altura. Os processos devem estar ativos apenas quando for necessário, devendo bloquear sempre que têm que esperar por algum evento.

O objetivo principal do trabalho passa pela compreensão dos mecanismos associados à execução e sincronização de processos e threads.

De maneira a **compilar e executar** o programa, executar dentro da diretoria **src** o comando **make all**, e depois dentro da diretoria **run** o comando **./probSemSharedMemAirLift.**

De notar que, de forma a conseguir correr o programa várias vezes seguidas, executou-se o comando **./clean.sh,** onde se teve que alterar as chaves predefinidas para as chaves do semáforo e memória partilhada. Para obter as chaves correu-se o comando **ipcs -m**.

# O Problema

## Descrição

O problema deste projeto é o seguinte: admite-se que é necessário transportar N passageiros entre as cidades *Origin* e *Target* usando uma ponte aérea de 1 avião. Os passageiros chegam ao aeroporto em tempos aleatórios e as regras são que o avião deve descolar de Origin sempre que:

- Está cheio;
- Já tem o número mínimo de passageiros e a fila está vazia;
- Quando todos os N passageiros já embarcaram.

A controlar o embarque está uma hospedeira que trata da verificação de identidade dos passageiros, dando autorização a cada passageiro para sair da fila de espera e entrar no avião. O piloto informa a hospedeira sempre que o avião está pronto para iniciar o processo de embarque, é informado por esta sempre que o *boarding* está completo e dá permissão aos passageiros para saírem do avião em *Target.* O piloto apenas inicia o voo de regresso quando o último passageiro sai do avião.

## Considerações

De modo a melhor compreensão do problema e resolução do mesmo torna-se fundamental perceber como é que os semáforos funcionam e como aplicá-los neste problema.

Dado que irão existir vários processos, que partilham as mesmas variáveis, a correr ao mesmo tempo podemos cair num problema de condição de corrida. De maneira a resolver isto é necessário haver algum tipo de **sincronização.** Para que a sincronização ocorra sem problemas, existe um conjunto de considerações que devemos ter:

- Enquanto não forem cumpridas uma das 3 condições abordadas anteriormente, o avião não vai levantar voo;
- O piloto tem de receber uma notificação a informar que *boarding* está completo e só depois é que pode iniciar o voo;
- O piloto apenas inicia o voo de regresso quando o último passageiro sair do avião;
- Enquanto estiverem no voo, os passageiros devem esperar que o voo termine;

No código fonte fornecido para a realização desta tarefa já se encontram definidos os semáforos, variáveis gerais e estados.

# Variáveis gerais e Estados

No ficheiro *probConst.h* estão algumas variáveis que devem ser usadas ao longo da resolução do problema, assim como os estados possíveis das 3 entidades do problema.

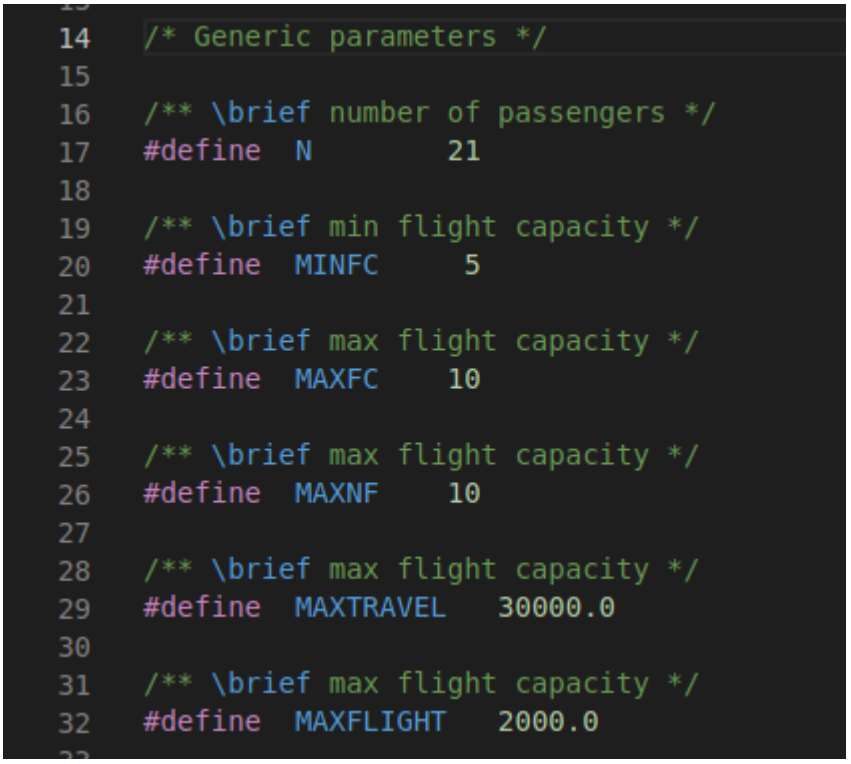
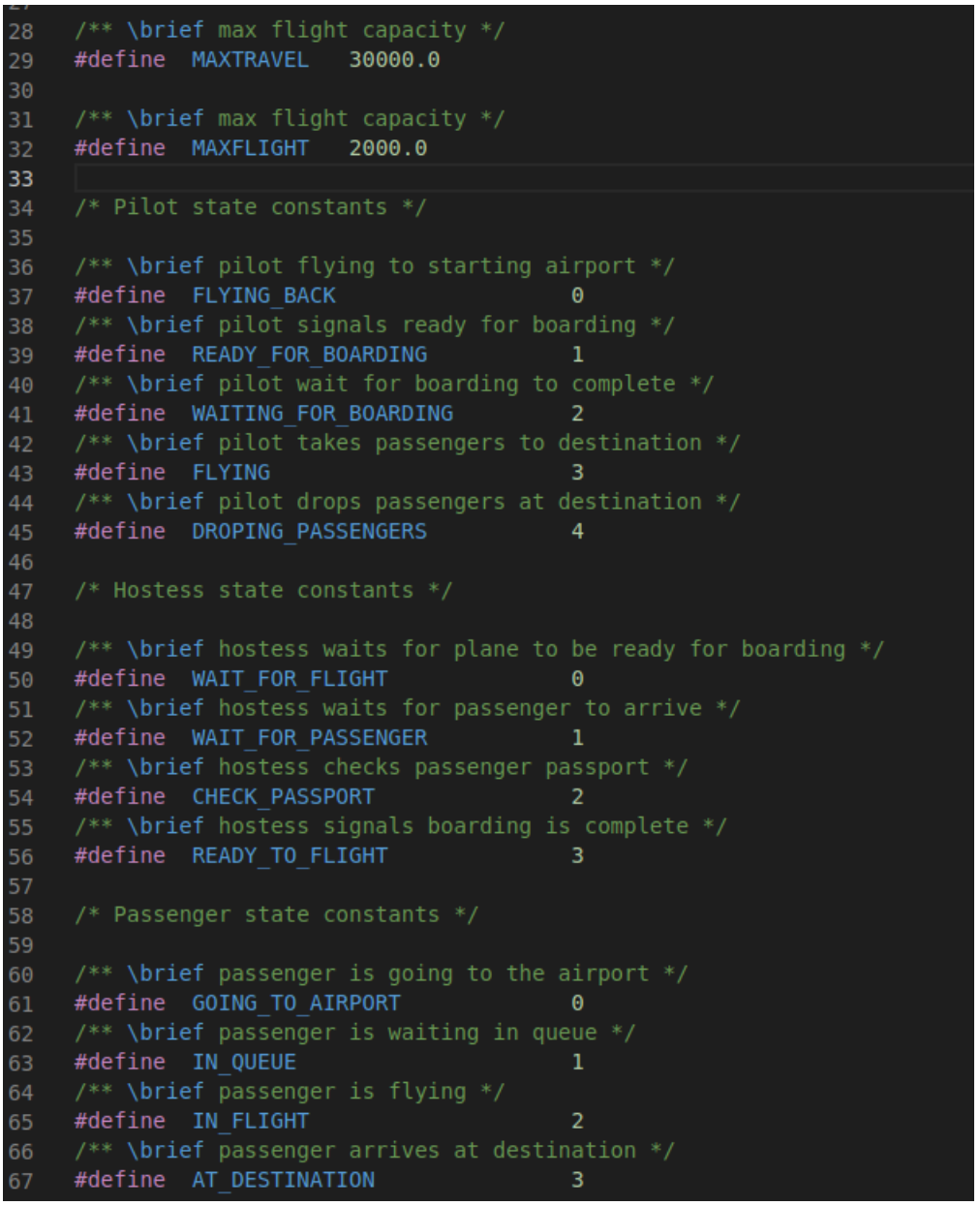


Fig.1 - Variáveis gerais do problema

Aqui estão definidas as variáveis globais a serem usadas, como por exemplo **N**, que representa o número de passageiros, **MINFC**, que representa o número mínimo de passageiros de um voo, **MAXFC,** representado o número máximo de passageiros num voo, etc.



```
28  /** \brief max flight capacity */
29  #define  MAXTRAVEL   30000.0
30
31  /** \brief max flight capacity */
32  #define  MAXFLIGHT   2000.0
33
34  /* Pilot state constants */
35
36  /** \brief pilot flying to starting airport */
37  #define  FLYING_BACK                0
38  /** \brief pilot signals ready for boarding */
39  #define  READY_FOR_BOARDING         1
40  /** \brief pilot wait for boarding to complete */
41  #define  WAITING_FOR_BOARDING       2
42  /** \brief pilot takes passengers to destination */
43  #define  FLYING                     3
44  /** \brief pilot drops passengers at destination */
45  #define  DROPING_PASSENGERS         4
46
47  /* Hostess state constants */
48
49  /** \brief hostess waits for plane to be ready for boarding */
50  #define  WAIT_FOR_FLIGHT            0
51  /** \brief hostess waits for passenger to arrive */
52  #define  WAIT_FOR_PASSENGER         1
53  /** \brief hostess checks passenger passport */
54  #define  CHECK_PASSPORT             2
55  /** \brief hostess signals boarding is complete */
56  #define  READY_TO_FLIGHT            3
57
58  /* Passenger state constants */
59
60  /** \brief passenger is going to the airport */
61  #define  GOING_TO_AIRPORT           0
62  /** \brief passenger is waiting in queue */
63  #define  IN_QUEUE                   1
64  /** \brief passenger is flying */
65  #define  IN_FLIGHT                  2
66  /** \brief passenger arrives at destination */
67  #define  AT_DESTINATION             3
```

Fig.2 - Estados possíveis das 3 entidades

Nesta figura podemos ver todos os estados possíveis que as 3 entidades podem tomar. O piloto pode ter 5 estados possíveis: **FLYING_BACK** (a voar até *ORIGIN*), **READY_FOR_BOARDING** (sinaliza a hospedeira que o *boarding* pode começar), **WAITING_FOR_BOARDING** (piloto espera que o *boarding* esteja completo), **FLYING** (a

voar até *TARGET*) e **DROPING_PASSENGERS** (deixar os passageiros no destino).

No que toca à hospedeira, esta pode ter 4 estados: **WAIT_FOR_FLIGHT** (espera que o piloto sinalize que o *boarding* pode começar), **WAIT_FOR_PASSENGER** (espera que os passageiros cheguem ao aeroporto), **CHECK_PASSPORT** (verifica o passaporte do passageiro) e **READY_TO_FLIGHT** (sinaliza o piloto que o *boarding* está feito).

Por fim, o passageiro também pode estar em 4 estados: **GOING_TO_AIRPORT** (está a deslocar-se para o aeroporto), **IN_QUEUE** (está à espera na fila), **IN_FLIGHT** (está no voo) e **AT_DESTINATION** (chegou ao destino).

# Estruturas de Dados

Um ficheiro que é igualmente muito importante e ao qual temos acesso no código fonte providenciado é o *probDataStruct.h*. Neste ficheiro podemos encontrar as **estruturas de dados** essenciais para o projeto.

O tipo STAT é uma estrutura que vai armazenar todos os estados das entidades do projeto. Neste projeto, temos presentes 3 parâmetros.

```
21  /**
22   *  \brief Definition of <em>state of the intervening entities</em> data type.
23   */
24  typedef struct
25  { /** \brief pilot state */
26      unsigned int pilotStat;
27      /** \brief hostess state */
28      unsigned int hostessStat;
29      /** \brief passengers state array */
30      unsigned int passengerStat[N];
31
32  } STAT;
```

Fig. 3 - Definição do STAT do tipo de dados das entidades intervenientes

O ***pilotStat*** e o ***hostessStat*** vão ser ambos um inteiro que vão guardar o estado do único piloto e única hospedeira, respetivamente, presentes no problema. O ***passengerStat[N]*** é um array de inteiros usado para os passageiros com tamanho igual ao número máximo de passageiros possíveis.

Outro tipo de dados presentes neste ficheiro e igualmente muito importante são os *FULL_STAT*. Estes vão guardar informação relativa a todas as variáveis do problema, variáveis essas que irão ser usadas ao longo da resolução do problema, algumas até sendo incrementadas ou decrementadas.

```
35  /**
36   *  \brief Definition of <em>full state of the problem</em> data type.
37   */
38  typedef struct
39  { /** \brief state of all intervening entities */
40      STAT st;
41      /** \brief number of passengers at each flight */
42      unsigned int nPassengersInFlight[MAXNF];
43      /** \brief flight number */
44      unsigned int nFlight;
45
46      /** \brief number of passengers waiting */
47      unsigned int nPassInQueue;
48      /** \brief number of passengers flying */
49      unsigned int nPassInFlight;
50      /** \brief total number of passengers already boarded in every flight */
51      unsigned int totalPassBoarded;
52      /** \brief air lift finished */
53      bool finished;
54      /** \brief passenger id of last passenger to check passport */
55      int passengerChecked;
56
57  } FULL_STAT;
```

Fig. 4 - *FULL_STAT* do tipo de dados do problema

Este tipo de dados vai guardar a estrutura de dados *STAT* na variável *st* e contém informação sobre:

- ***nPassengersInFlight[MAXNF]*** - número de passageiros em cada voo;
- ***nFlight*** - número do voo;
- ***nPassInQueue*** - número de passageiros à espera (na fila);
- ***nPassInFlight*** - número de passageiros a voar;
- ***totalPassBoarded*** - número de passageiros que já fizeram o *boarding;*
- ***finished*** - true/false consoante se o voo já terminou;
- ***passengerChecked*** - id do último passageiro a quem foi verificado o passaporte.

# Semáforos Pré - Definidos

De maneira a ajudar na resolução do problema, existem, no ficheiro *sharedDataSync.h,* semáforos já criados pelo professor e prontos a serem usados na resolução do problema. Como já referido anteriormente, os semáforos irão ser muito importantes no que toca a resolver problemas de sincronização, visto que permitem controlar os processos sequencialmente.

```
typedef struct
        { /** \brief full state of the problem */
          FULL_STAT fSt;

          /* semaphores ids */
          /** \brief identification of critical region protection semaphore – val = 1 */
          unsigned int mutex;
          /** \brief identification of semaphore used by hostess to wait for passengers - val = 0 */
          unsigned int passengersInQueue;
          /** \brief identification of semaphore used by passengers to wait for hostess – val = 0 */
          unsigned int passengersWaitInQueue;
          /** \brief identification of semaphore used by passengers to wait for flight to end – val = 0 */
          unsigned int passengersWaitInFlight;
          /** \brief identification of semaphore used by hostess to wait for starting boarding – val = 0  */
          unsigned int readyForBoarding;
          /** \brief identification of semaphore used by pilot to wait for boarding to complete - val = 0 */
          unsigned int readyToFlight;
          /** \brief identification of semaphore used by hostess to wait for passenger identification - val = 0 */
          unsigned int idShown;
          /** \brief identification of semaphore used by pilot to wait for last passenger to leave plane - val = 0 */
          unsigned int planeEmpty;

        } SHARED_DATA;
```

Fig.5 - Semáforos

Os semáforos aqui apresentados são:

- ***mutex*** - usado para bloquear qualquer outro processo que tente entrar numa região, protegendo a execução do processo atual;
- ***passengersInQueue*** **-** usado pela hospedeira para esperar pelos passageiros na fila;
- ***passengersWaitInQueue*** **-** semáforo usado pelos passageiros para esperar pela hostess enquanto estão na fila

- ***passengersWaitInFlight*** - semáforo utilizado pelos passageiros para esperar que o voo termine;
- ***readyForBoarding*** - usado pela hospedeira para esperar que o piloto dê luz verde para o *boarding* começar;
- ***readyToFlight*** - usado pelo piloto para esperar que o *boarding* esteja completo;
- ***idShown*** - usado pela hospedeira para esperar que o passageiro mostre o id;
- ***planeEmpty*** - usado pelo piloto para esperar que o último passageiro abandone o avião.

De modo a auxiliar na implementação dos semáforos e perceber em que partes do código iriam ser implementados desenhou-se a seguinte tabela:

**Tabela de Semáforos**

| Semáforo | Entidade Down | Função Down | Entidade Up | Função Up |
|---|---|---|---|---|
| passengersInQueue | Hostess | waitForPassenger | Passengers | waitInQueue |
| passengersWaitInQueue | Passengers | waitInQueue | Hostess | checkPassport |
| passengersWaitInFlight | Passengers | waitUntilDestination | Pilot | dropPassengersAtTarget |
| readyForBoarding | Hostess | waitForNextFlight | Pilot | signalReadyForBoarding |
| readyToFlight | Pilot | waitUntilReadytoFlight | Hostess | signalReadyToFlight |
| idShown | Hostess | checkPassport | Passengers | waitInQueue |
| planeEmpty | Pilot | dropPassengersAtTarget | Passengers | waitUntilDestination |

Fig.6 - Tabela de Semáforos criada para auxiliar no projeto

# Implementação

## Piloto

No que toca à implementação da solução, começou-se por desenvolver o código do piloto, não só por parecer ser o mais fácil de implementar primeiro, mas também visto ser a entidade que envolve o menor número de semáforos.

### *Função flight (bool go)*

```
137 static void flight (bool go)
138 {
139     if (semDown (semgid, sh->mutex) == -1) {                                    /* enter critical region */
140         perror ("error on the up operation for semaphore access (PT)");
141         exit (EXIT_FAILURE);
142     }
143 
144     /* insert your code here */
145 
146     if (go == true) {                       //caso seja true
147         sh->fSt.st.pilotStat = FLYING;      //caso seja true significa que o piloto está a viajar de origin para target
148         saveState(nFic, &sh->fSt);
149     }else{
150         sh->fSt.st.pilotStat = FLYING_BACK;     //caso contrário está a está a viajar de target para origin
151         saveState(nFic, &sh->fSt);
152     }
153 
154     if (semUp (semgid, sh->mutex) == -1) {                                      /* exit critical region */
155         perror ("error on the up operation for semaphore access (PT)");
156         exit (EXIT_FAILURE);
157     }
158 
159     usleep((unsigned int) floor ((MAXFLIGHT * random ()) / RAND_MAX + 100.0));
160 }
```

Fig. 7 - Função flight(bool go)

Nesta função, criou-se um ciclo *if* que toma o valor da variável *go,* caso *go* retorne *true* então significa que o piloto está a viajar de *Origin* para *Target.* Caso contrário, significa que está a viajar no sentido contrário.

Em ambos os casos definiu-se o estado inicial do piloto, sendo *FLYING* se *go* for *true*, e *FLYING_BACK* caso contrário.

## *Função signalReadyForBoarding()*

```
170  static void signalReadyForBoarding ()
171  {
172      if (semDown (semgid, sh->mutex) == -1) {                                        /* enter critical region */
173          perror ("error on the up operation for semaphore access (PT)");
174          exit (EXIT_FAILURE);
175      }
176
177      /* insert your code here */
178
179      sh->fSt.st.pilotStat = READY_FOR_BOARDING;                  //piloto atualiza o seu estado, sinalizando que o avião está pronto para board
180      saveState(nFic, &sh->fSt);                                  //guardar estado inicial
181      sh->fSt.nFlight++;                                          //atualiza o número do voo
182      saveStartBoarding(nFic, &sh->fSt);                          //guardar estado inicial do logging num ficheiro
183
184      if (semUp (semgid, sh->mutex) == -1) {                                          /* exit critical region */
185          perror ("error on the up operation for semaphore access (PT)");
186          exit (EXIT_FAILURE);
187      }
188
189      /* insert your code here */
190
191      if (semUp(semgid, sh->readyForBoarding) == -1) {            //piloto notifica hostess que o boarding pode começar
192          perror("error on the down operation for semaphore access (PT)");
193          exit(EXIT_FAILURE);
194      }
195  }
```

Fig. 8 - Função *signalReadyForBoarding()*

De seguida, o piloto vai sinalizar que o boarding pode começar, atualizando o seu estado inicial para *READY_FOR_BOARDING*.

O número de voo é incrementado, sinalizando que este já se trata de outro voo, visto que o avião está em fase de "preenchimento" e vai partir para outra viagem.

Quanto aos semáforos, o semáforo *readyForBoarding* incrementa uma unidade, desbloqueando a hospedeira que estava bloqueada e que pode agora começar a efetuar o *boarding*.

## *Função waitUntilReadyToFlight()*

```
204  static void waitUntilReadyToFlight ()
205  {
206      if (semDown (semgid, sh->mutex) == -1) {                                                    /* enter critical region */
207          perror ("error on the up operation for semaphore access (PT)");
208          exit (EXIT_FAILURE);
209      }
210
211      /* insert your code here */
212
213      sh->fSt.st.pilotStat = WAITING_FOR_BOARDING;               //piloto atualiza o seu estado para WAITING_FOR_BOARDING, sinalizando que està à espera dos
214      saveState(nFic, &sh->fSt);
215
216
217      if (semUp (semgid, sh->mutex) == -1) {                                                      /* exit critical region */
218          perror ("error on the up operation for semaphore access (PT)");
219          exit (EXIT_FAILURE);
220      }
221
222      /* insert your code here */
223      if (semDown(semgid, sh->readyToFlight) == -1) {            //semáforo utilizado pelo piloto para esperar que o boarding esteja completo
224          perror ("error on the down operation for semaphore access (PT)");
225          exit (EXIT_FAILURE);
226      }
```

Fig. 9 - Função *waitUntilReadyToFlight()*

Passando agora à função *waitUntilReadyToFlight()*. O piloto espera que o boarding esteja completo, atualizando assim o seu estado para *WAITING_FOR_BOARDING*.

De seguida, bloqueou-se o semáforo *readyToFlight,* utilizado pelo piloto para esperar que o *boarding* esteja completo, simbolizando que o piloto vai ficar bloqueado até o boarding estar completo.

## *Função dropPassengersAtTarget()*

```
238  static void dropPassengersAtTarget ()
239  {
240      if (semDown (semgid, sh->mutex) == -1) {                                                /* enter critical region */
241          perror ("error on the down operation for semaphore access (PT)");
242          exit (EXIT_FAILURE);
243      }
244
245      /* insert your code here */
246
247      saveFlightArrived(nFic, &sh->fSt);
248      sh->fSt.st.pilotStat = DROPING_PASSENGERS;             //piloto atualiza o seu estado para DROPING_PASSENGERS, sinalizando que está a deixar os passageiros sairem
249      saveState(nFic, &sh->fSt);
250
251      if (semUp (semgid, sh->mutex) == -1)  {                                                 /* exit critical region */
252          perror ("error on the up operation for semaphore access (PT)");
253          exit (EXIT_FAILURE);
254      }
255
256      /* insert your code here */
257      for (int i = 0; i < sh->fSt.nPassengersInFlight[sh->fSt.nFlight-1]; i++){   //ciclo for que percorrer o número de passageiros no avião
258          if (semUp(semgid, sh->passengersWaitInFlight) == -1){                   //semáforo usado pelos passageiros para esperar pelo piloto, vai estar up
259              perror("error on the up operation for semaphore access (PT)");
260              exit(EXIT_FAILURE);
261          }
262      }
263
264      if (semDown (semgid, sh->planeEmpty) == -1) {                       //semáforo utilizado pelo piloto para esperar pelo último passageiro para sair do avião
265          perror ("error on the down operation for semaphore access (PT)");
266          exit (EXIT_FAILURE);
267      }
268
269
270      if (semDown (semgid, sh->mutex) == -1) {                                                /* enter critical region */
271          perror ("error on the down operation for semaphore access (PT)");
272          exit (EXIT_FAILURE);
273      }
274
275      /* insert your code here */
276      sh->fSt.st.pilotStat = FLYING_BACK;             //piloto atualiza o seu estado para FLYING_BACK, sinalizando que está a regressar após deixar os passageiros
277      saveState(nFic, &sh->fSt);
278      saveFlightReturning(nFic, &sh->fSt);
279
280      if (semUp (semgid, sh->mutex) == -1)  {                                                 /* exit critical region */
281          perror ("error on the up operation for semaphore access (PT)");
282          exit (EXIT_FAILURE);
283      }
```

Quanto à última função do ficheiro relativo ao piloto, *dropPassengersAtTarget(),* esta é referente ao piloto deixar os passageiros no destino.

Ao chegar, o piloto deixa os passageiros no destino e vai atualizar o seu estado inicial para *DROPING_PASSENGERS*.

De seguida, incrementa-se o valor do semáforo *passengersWaitInFlight* para cada passageiro de cada voo, libertando assim todos os passageiros que estavam bloqueados, de maneira a estes poderem desembarcar.

Depois, decrementa-se o valor do semáforo *planeEmpty,* semáforo usado pelo piloto para esperar que o último passageiro saia, de modo a bloquear o piloto até que o último passageiro saia do avião, impedindo-o de voltar a voar sem que o último passageiro saia.

Por fim, assim que o último passageiro saia do avião, o piloto vai atualizar e guardar o seu estado inicial para *FLYING_BACK e* é impressa a mensagem informativa que o avião está a voltar para ORIGIN.

## Hospedeira

O próximo código a ser analisado foi o da segunda entidade do problema, a hospedeira.

### *Função waitForNextFlight()*

```
142  static void waitForNextFlight ()
143  {
144      if (semDown (semgid, sh->mutex) == -1)  {                                          /* enter critical region */
145          perror ("error on the up operation for semaphore access (HT)");
146          exit (EXIT_FAILURE);
147      }
148
149      /* insert your code here */
150      sh->fSt.st.hostessStat = WAIT_FOR_FLIGHT;       //hostess atualiza o seu internal state para WAIT_FOR_FLIGHT
151      saveState(nFic, &sh->fSt);          //guardar internal state
152
153
154      if (semUp (semgid, sh->mutex) == -1)                                              /* exit critical region */
155      { perror ("error on the down operation for semaphore access (HT)");
156          exit (EXIT_FAILURE);
157      }
158
159      /* insert your code here */
160      if (semDown(semgid, sh->readyForBoarding) == -1) {         //semáforo utilizado pela hostess para esperar que o boarding comece
161          perror("error on the down operation for semaphore access (HT)");
162          exit(EXIT_FAILURE);
163      }
164
165  }
```

Fig. 11 - Função waitForNextFlight()

A primeira função presente no código é a função *waitForNextFlight(),* tendo como descrição esperar pelo próximo voo. Esta função define o estado inicial da hospedeira como sendo WAIT_FOR_FLIGHT, tendo assim o valor 0, o que significa que ainda está à espera que o avião esteja pronto para *boarding*.

O semáforo *readyForBoarding* será decrementado, bloqueando assim a hospedeira, que vai ficar à espera de poder começar o *boarding*.

### *Função waitForPassenger()*

```
174 static void waitForPassenger ()
175 {
176     if (semDown (semgid, sh->mutex) == -1)                                          /* enter critical region */
177     { perror ("error on the up operation for semaphore access (HT)");
178         exit (EXIT_FAILURE);
179     }
180 
181     /* insert your code here */
182     sh->fSt.st.hostessStat = WAIT_FOR_PASSENGER;        //hostess espera que o passageiro chegue ao aeroporto e atualiza o seu estado para WAIT_FOR_PASSENGER
183     saveState(nFic, &sh->fSt);      //guardar o internal state
184 
185     if (semUp (semgid, sh->mutex) == -1) {                                          /* exit critical region */
186      perror ("error on the down operation for semaphore access (HT)");
187         exit (EXIT_FAILURE);
188     }
189 
190     /* insert your code here */
191     if (semDown(semgid, sh->passengersInQueue) == -1){          //semáforo usado pela hostess para esperar pelos passageiros
192         perror("error on the down operation for semaphore access (HT)");
193         exit(EXIT_FAILURE);
194     }
195 }
196 
```

Fig. 12 - Função waitForPassenger()

Nesta função a hospedeira espera que os passageiros cheguem ao aeroporto.

Portanto, atualiza-se o estado da hospedeira para *WAIT_FOR_PASSENGER* e guarda-se.

Quanto ao semáforo *passengersInQueue,* este será decrementado bloqueando assim a hospedeira, que vai ter de esperar até que os passageiros cheguem.

### *Função checkPassport()*

Esta função tem como objetivo a hospedeira verificar o passaporte do passageiro e esperar que este lhe mostre o id.

```
210 static bool checkPassport()
211 {
212     bool last;
213 
214     /* insert your code here */
215     if (semUp(semgid, sh->passengersWaitInQueue) == -1) {                //semáforo usado pelos passageiros para esperar pela hostess vai estar up aqui visto que a
216         perror ("error on the down operation for semaphore access (HT)");
217         exit (EXIT_FAILURE);
218         }
219 
220     if (semDown (semgid, sh->mutex) == -1) {                                                    /* enter critical region */
221         perror ("error on the up operation for semaphore access (HT)");
222         exit (EXIT_FAILURE);
223 
224     }
225 
226     /* insert your code here */
227     sh->fSt.st.hostessStat = CHECK_PASSPORT;               //hostess atualiza o seu estado para CHECK_PASSPORT
228     saveState(nFic, &sh->fSt);      //primeiro save internal state
229 
230     if (semUp (semgid, sh->mutex) == -1)     {                                                  /* exit critical region */
231         perror ("error on the up operation for semaphore access (HT)");
232         exit (EXIT_FAILURE);
233     }
234 
235     /* insert your code here */
236     if (semDown (semgid, sh->idShown) == -1) {                     //semáforo utilizado pela hostess para esperar que os passageiros mostrem o seu id
237         perror ("error on the up operation for semaphore access (HT)");
238         exit (EXIT_FAILURE);
239     }
240 
241     if (semDown (semgid, sh->mutex) == -1)  {                                                  /* enter critical region */
242         perror ("error on the up operation for semaphore access (HT)");
243         exit (EXIT_FAILURE);
244 
247     /* insert your code here */
248     savePassengerChecked(nFic, &sh->fSt);   //guardar que o passageiro já foi checked
249     sh->fSt.nPassInQueue--;     //passageiro deixa de estar na fila, número de passageiros na fila é decrementado
250     sh->fSt.totalPassBoarded++; //passageiro deixa de estar na fila e entra no avião, número de passageiros boarded é incrementado
251     sh->fSt.nPassInFlight++;    //passageiros que já estão boarded também vão estar no avião
252     saveState(nFic, &sh->fSt);
253 
254     if (nPassengersInFlight() == MAXFC){
255         last = true;
256 
257     }else if (nPassengersInFlight() >= MINFC && nPassengersInQueue() == 0)
258     {
259         last = true;
260 
261     }else if (sh->fSt.totalPassBoarded == N)
262     {
263         last = true;
264 
265     }else{
266 
267         last = false;
268         if (last == false){
269             sh->fSt.st.hostessStat = WAIT_FOR_PASSENGER;     //caso não seja o último passageiro a hostess vai continuar à espera de mais passageiros
270             saveState(nFic, &sh->fSt);      //segundo internal state
271         }
272     }
273 
274     if (semUp (semgid, sh->mutex) == -1) {                                                      /* exit critical region */
275         perror ("error on the up operation for semaphore access (HT)");
276         exit (EXIT_FAILURE);
277     }
278 
279     /* insert your code here */
280 
281     return last;
282 }
```

Esta função vai retornar *true* quando o último passageiro dar o seu passaporte para a hospedeira verificar. O último passageiro é determinado da seguinte forma:

- Voo está na capacidade máxima;

- A capacidade do voo está igual ou maior à capacidade mínima que um voo pode ter e não há nenhum passageiro na fila;
- Não há mais passageiros.

Começa-se por incrementar o semáforo *passengersWaitInQueue,* de modo a que os passageiros sejam libertados e possam começar a mostrar o passaporte à hospedeira e ir para o avião.

Dado isto, a hospedeira atualiza o seu estado para *CHECK_PASSPORT*, e começa a verificar os passaportes dos passageiros.

O semáforo *idShown*, usado pela hospedeira para esperar pelos passaportes, vai ser decrementado, bloqueando assim a hospedeira, e fazendo-a esperar ser desbloqueada pelo incremento deste semáforo por parte dos passageiros.

Após o passageiro ter sido verificado, decrementa-se uma unidade à variável da estrutura FULL_STAT *nPassInQueue* uma vez que, quando é verificado, o passageiro sai da fila e vai para o avião.

Dado isto, incrementa-se uma unidade a duas variáveis da estrutura FULL_STAT: *nPassInFlight* e *totalPassBoarded.* A primeira é incrementada dado que o passageiro quando sai da fila vai para o avião, e a última é referente ao total de passageiros que já entraram no avião.

De seguida, e de maneira a determinar a variável *last* (último passageiro) criou-se um ciclo *for* que vai percorrer as 3 alternativas referentes anteriormente. Caso o número de passageiros no voo seja igual à capacidade máxima do avião, a variável *last* assume o valor *true*; caso a capacidade do voo seja igual ou maior à capacidade mínima que um voo pode ter e o número de passageiros na fila seja igual a zero, a variável *last* assume o valor *true*; por fim, se a variável definida anteriormente, *totalPassBoarded* seja igual a N (número de passageiros) a variável *last* assume o valor *true.*

Se nenhuma destas condições se verificar, a variável *last* assume o valor *false*, e o processo de verificação de passaportes continua. Assim sendo, a hospedeira atualiza o seu estado para *WAIT_FOR_PASSENGER*, e vai continuar a verificar passaportes.

## *Função signalReadyToFlight()*

Esta função tem como função sinalizar que o voo está pronto para partir

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

    /* insert your code here */
    sh->fSt.nPassengersInFlight[sh->fSt.nFlight-1] = nPassengersInFlight();     //regista o número de passageiros no voo
    sh->fSt.st.hostessStat = READY_TO_FLIGHT;                                   //atualiza o seu estado para READY_TO_FLIGHT
    saveState(nFic, &sh->fSt);
    saveFlightDeparted(nFic, &sh->fSt);

    if (sh->fSt.totalPassBoarded == N)          //caso o número total de passageiros boarded seja igual ao número de passageiros
        sh->fSt.finished = true;                //então o voo acabou

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

    /* insert your code here */
    if (semUp (semgid, sh->readyToFlight) == -1) {                 //semáforo usado para informar o piloto que o avião está pronto para partir
        perror ("error on the up operation for semaphore access (HT)");
        exit (EXIT_FAILURE);
    }
}
```

De seguida, vai verificar se o total de passageiros que já embarcaram é igual ao número de passageiros e, em caso positivo, a variável *finished* assume o valor *true,* o que significa que é o último voo.

Por fim, incrementa-se o semáforo *readyToFlight*, desbloqueando assim o piloto e permitindo-lhe descolar.

## Passageiro

Passando à última entidade do problema, o passageiro.

### *Função waitInQueue(passengerId)*

Esta função tem como objetivo o passageiro esperar pela sua vez para o passaporte ser verificado.

```
141  static void waitInQueue (unsigned int passengerId)
142  {
143      if (semDown (semgid, sh->mutex) == -1) {                                        /* enter critical region */
144          perror ("error on the down operation for semaphore access (PG)");
145          exit (EXIT_FAILURE);
146      }
147
148      /* insert your code here */
149
150      sh->fSt.nPassInQueue++;                 //passageiro atualiza o número de passageiros na fila
151      sh->fSt.st.passengerStat[passengerId] = IN_QUEUE;       //passageiro atualiza o seu estado para IN_QUEUE
152      saveState(nFic, &sh->fSt);              //primeiro save state
153
154      if (semUp (semgid, sh->mutex) == -1)                                            /* exit critical region */
155      { perror ("error on the up operation for semaphore access (PG)");
156          exit (EXIT_FAILURE);
157      }
158
159      /* insert your code here */
160      if (semUp (semgid, sh->passengersInQueue) == -1) {
161          perror ("error on the up operation for semaphore access (PG)");
162          exit (EXIT_FAILURE);
163      }
164
165      if (semDown (semgid, sh->passengersWaitInQueue) == -1) {
166          perror ("error on the up operation for semaphore access (PG)");
167          exit (EXIT_FAILURE);
168      }
169
170
171      if (semDown (semgid, sh->mutex) == -1) {                                        /* enter critical region */
172          perror ("error on the down operation for semaphore access (PG)");
173          exit (EXIT_FAILURE);
174      }
176      /* insert your code here */
177
178      sh->fSt.passengerChecked = passengerId;         //fazendo esta igualdade permite, na linha seguinte, assumir que os passageiros que a
179      sh ->fSt.st.passengerStat[passengerId] = IN_FLIGHT;   //segundo state update, aqui os passageiros que já foram verificados pela hoste
180      saveState(nFic, &sh->fSt);      //guardar internal state pela segunda vez
181      savePassengerChecked(nFic, &sh->fSt);
182
183
184      if (semUp (semgid, sh->mutex) == -1) {                                      /* enter critical region */
185          perror ("error on the down operation for semaphore access (PG)");
186          exit (EXIT_FAILURE);
187      }
188
189      /* insert your code here */
190
191      if (semUp (semgid, sh->idShown) == -1) {          //semáforo usado pela hostess para esperar que o passageiro lhe mostre o id, neste
192          perror ("error on the up operation for semaphore access (PG)");
193          exit (EXIT_FAILURE);
194      }
195  }
196
```

Primeiramente, dentro da região crítica, incrementa-se uma unidade à variável da estrutura *FULL_STAT nPassInQueue,* uma vez que o passageiro vai estar na fila e atualiza-se o estado do passageiro para *IN_QUEUE*, sinalizando que está na fila.

De seguida, desbloqueia-se as hospedeiras ao incrementar/fazer up do semáforo *passengersInQueue,* e decrementa-se o semáforo *passengersWaitInQueue,* bloqueando os passageiros, de modo a que eles esperem na fila.

Depois, atribui-se à variável *passengerChecked* o valor do *passengerId* dado que desta forma torna-se simples perceber qual o último passageiro a embarcar. Por fim, atualiza-se o estado do último passageiro a embarcar para IN_FLIGHT, sinalizando que já está a voar.

Por fim incrementa-se o semáforo idShown, desbloqueando desta maneira a hospedeira de forma a que esta possa começar a ver a identificação dos passageiros.

### *Função waitUntilDestination(passengerId)*

Esta função tem como objetivo os passageiros esperarem que o voo termine e chegar ao destino.

```
208 static void waitUntilDestination (unsigned int passengerId)
209 {
210
211     bool lastPassenger;         //último passageiro do avião
212
213     /* insert your code here */
214     if (semDown (semgid, sh->passengersWaitInFlight) == -1) {           //semáforo usado pelos passageiros para esperar que o voo termine
215         perror ("error on the down operation for semaphore access (PG)");
216         exit (EXIT_FAILURE);
217     }
218
219
220     if (semDown (semgid, sh->mutex) == -1) {                                            /* enter critical region */
221         perror ("error on the down operation for semaphore access (PG)");
222         exit (EXIT_FAILURE);
223     }
224
225     /* insert your code here */
226     sh->fSt.nPassInFlight--;        //ao chegar ao destino o passageiro atualiza o número de passageiros do avião
227     sh->fSt.st.passengerStat[passengerId] = AT_DESTINATION;
228     saveState(nFic, &sh->fSt);
229
230     if (sh->fSt.nPassInFlight == 0){
231         lastPassenger = true;
232         if (lastPassenger == true){
233             if (semUp(semgid, sh->planeEmpty) == -1) {          //semáforo utilizado pelo piloto para esperar que o avião esteja vazio, neste caso vai estar up porque
234             perror ("error on the down operation for semaphore access (PG)");
235             exit (EXIT_FAILURE);
236             }
237         }
238
239     }
240
241     if (semUp (semgid, sh->mutex) == -1) {                                              /* enter critical region */
242         perror ("error on the down operation for semaphore access (PG)");
243         exit (EXIT_FAILURE);
```

Fig.16 - Função waitUntilDestination(passengerId)

Primeiramente, decrementa-se o semáforo *passengersWaitInFlight,* bloqueando os passageiros e fazê-los esperar no avião.

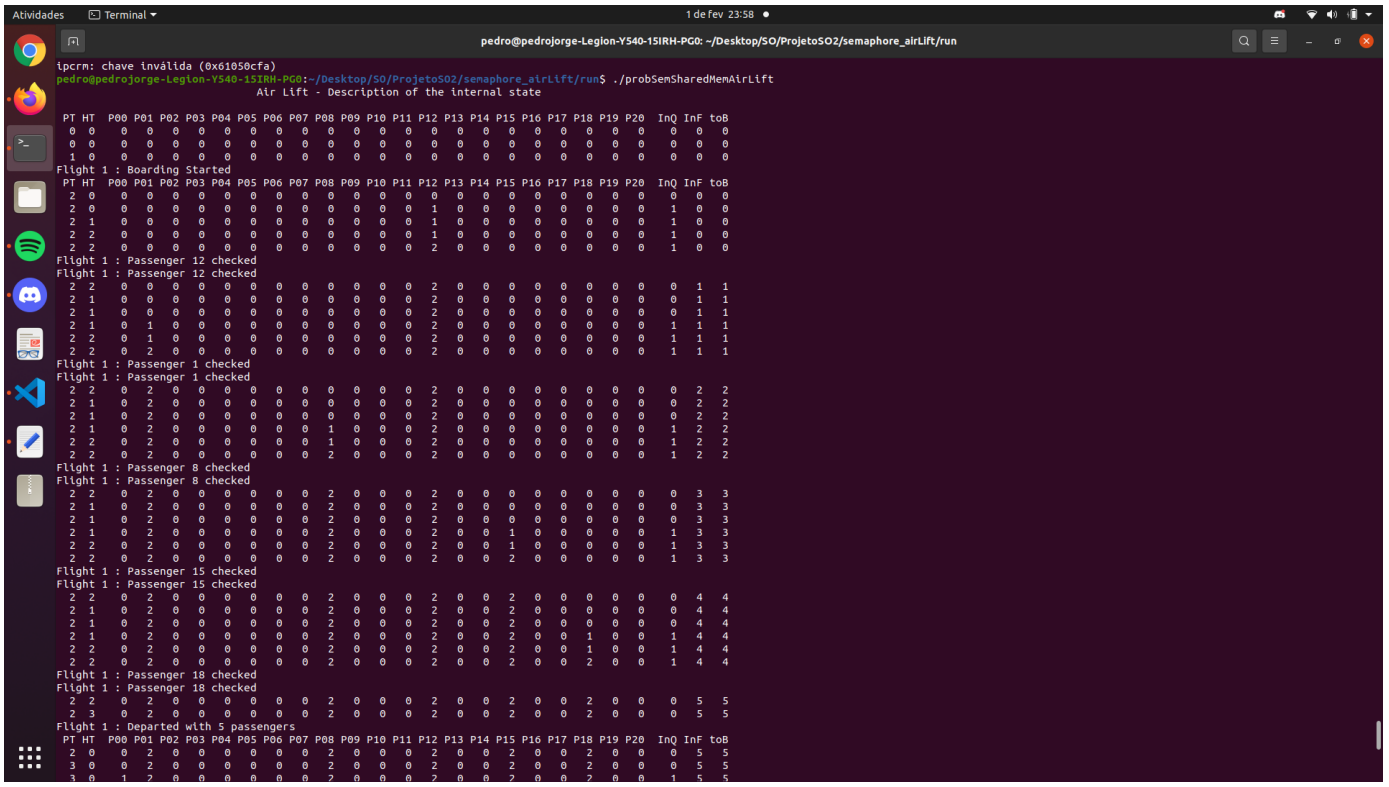
De seguida, ao entrar na região crítica, decrementa-se uma unidade à variável da estrutura *FULL_STAT nPassInFlight,* simbolizando que o passageiro está a sair do avião. Depois, atualiza-se o estado inicial desses passageiros para *AT_DESTINATION*, simbolizando que já chegaram ao destino.

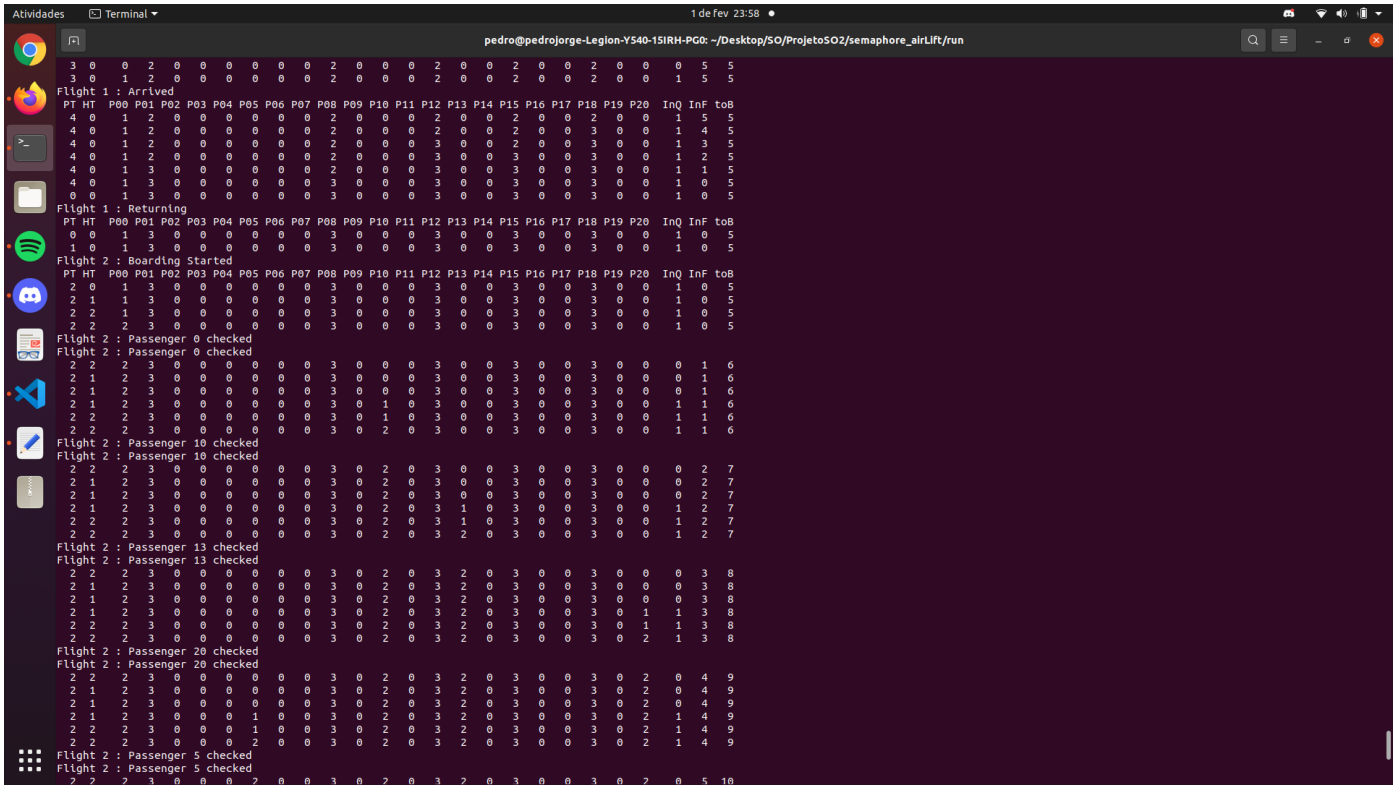
Por fim, e após ter sido criada a variável *bool lastPassenger*, representativa do último passageiro, criou-se uma condição if que vai verificar se o número de passageiros no voo é igual a

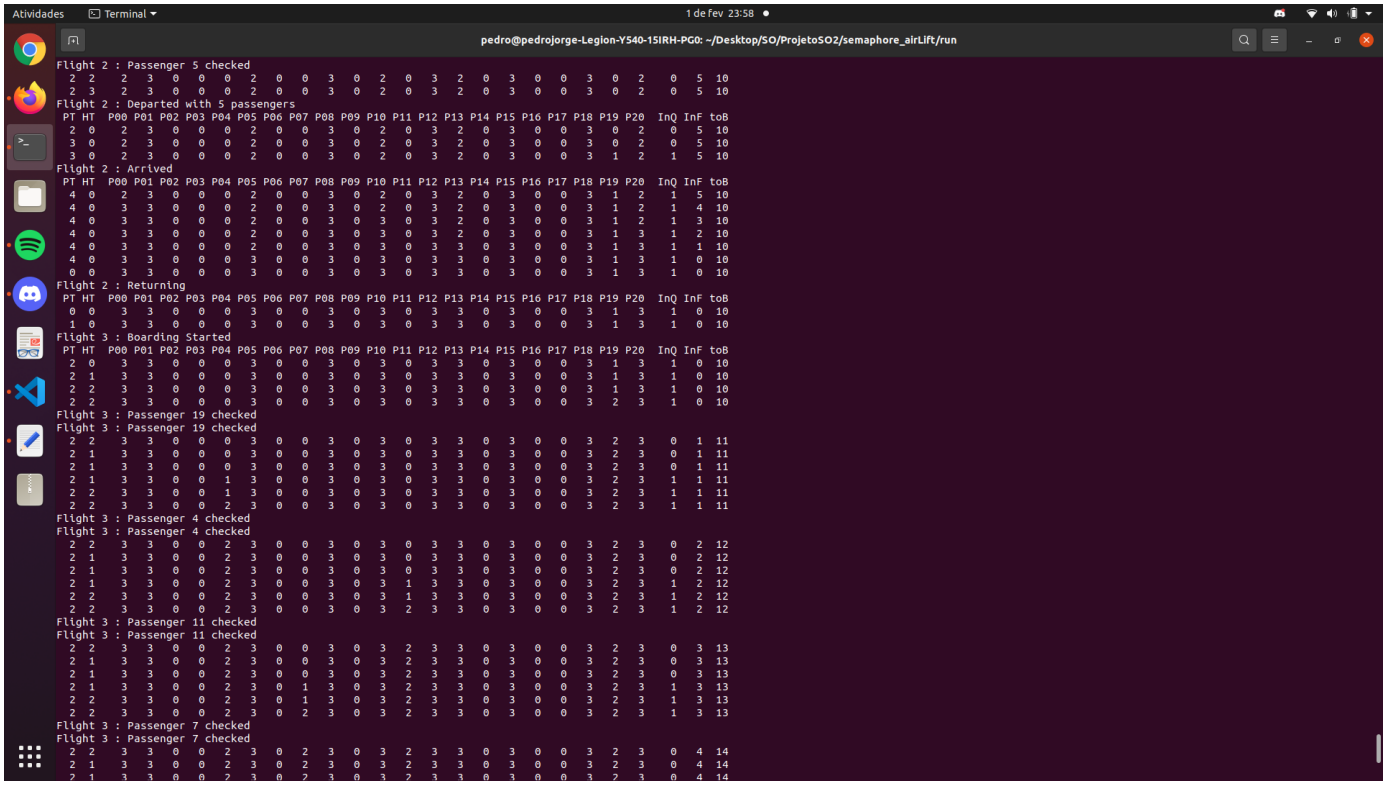
zero. Em caso positivo, a variável *lastPassenger* assume o valor de *true* e incrementa-se o semáforo planeEmpty, desbloqueando assim o piloto, permitindo-o voltar para *ORIGIN*.

# Resultados Obtidos

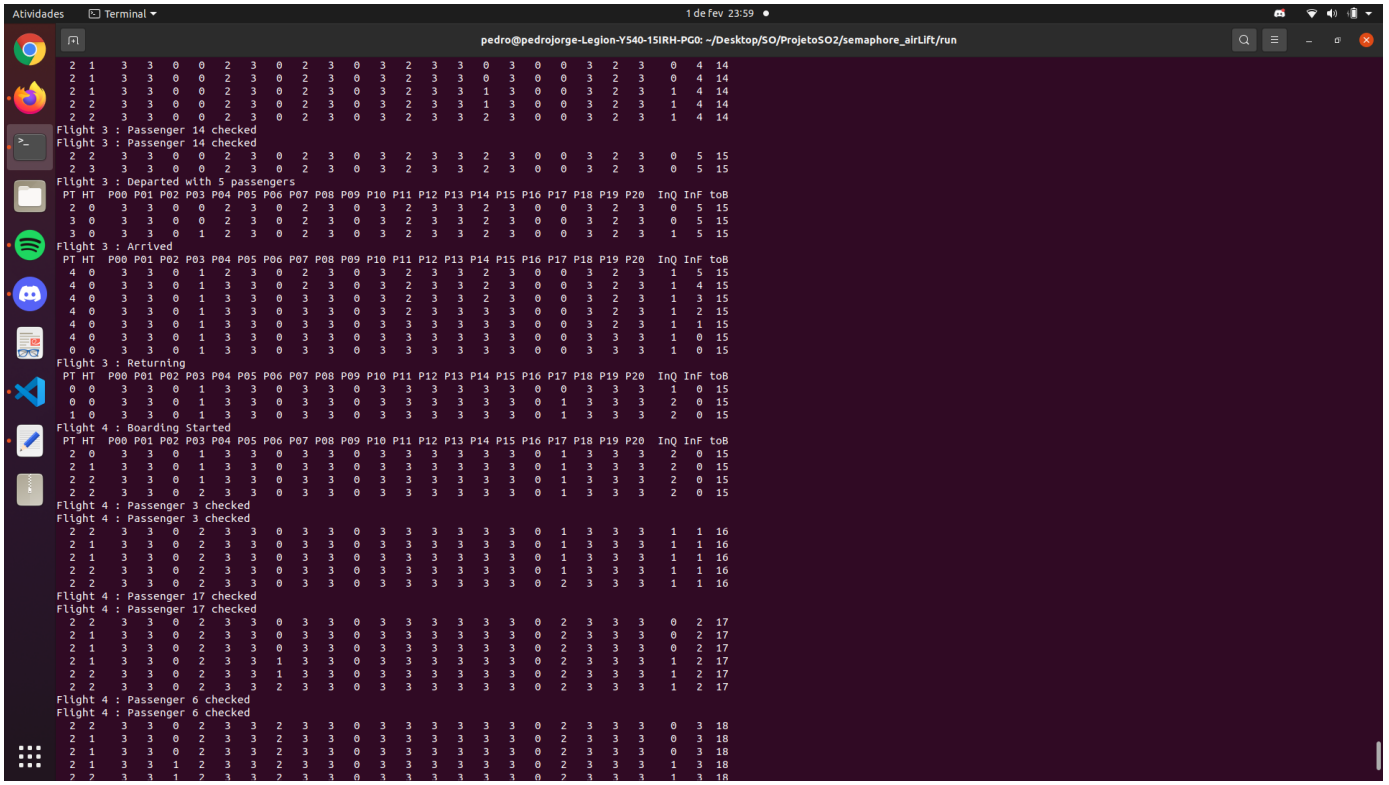
Após esta implementação descrita acima, e fazendo os comandos referidos no início deste relatório, estes foram os resultados obtidos:



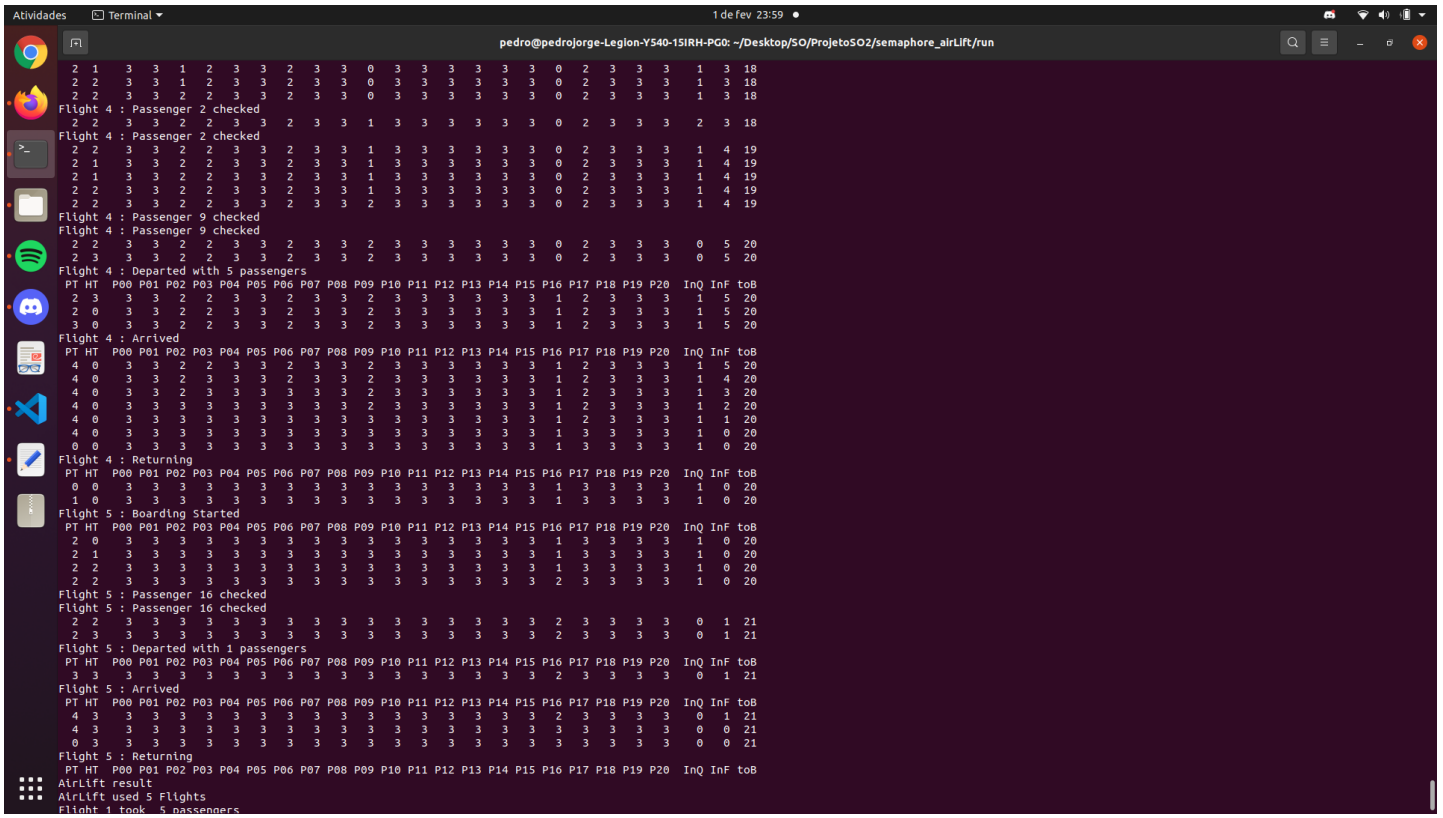



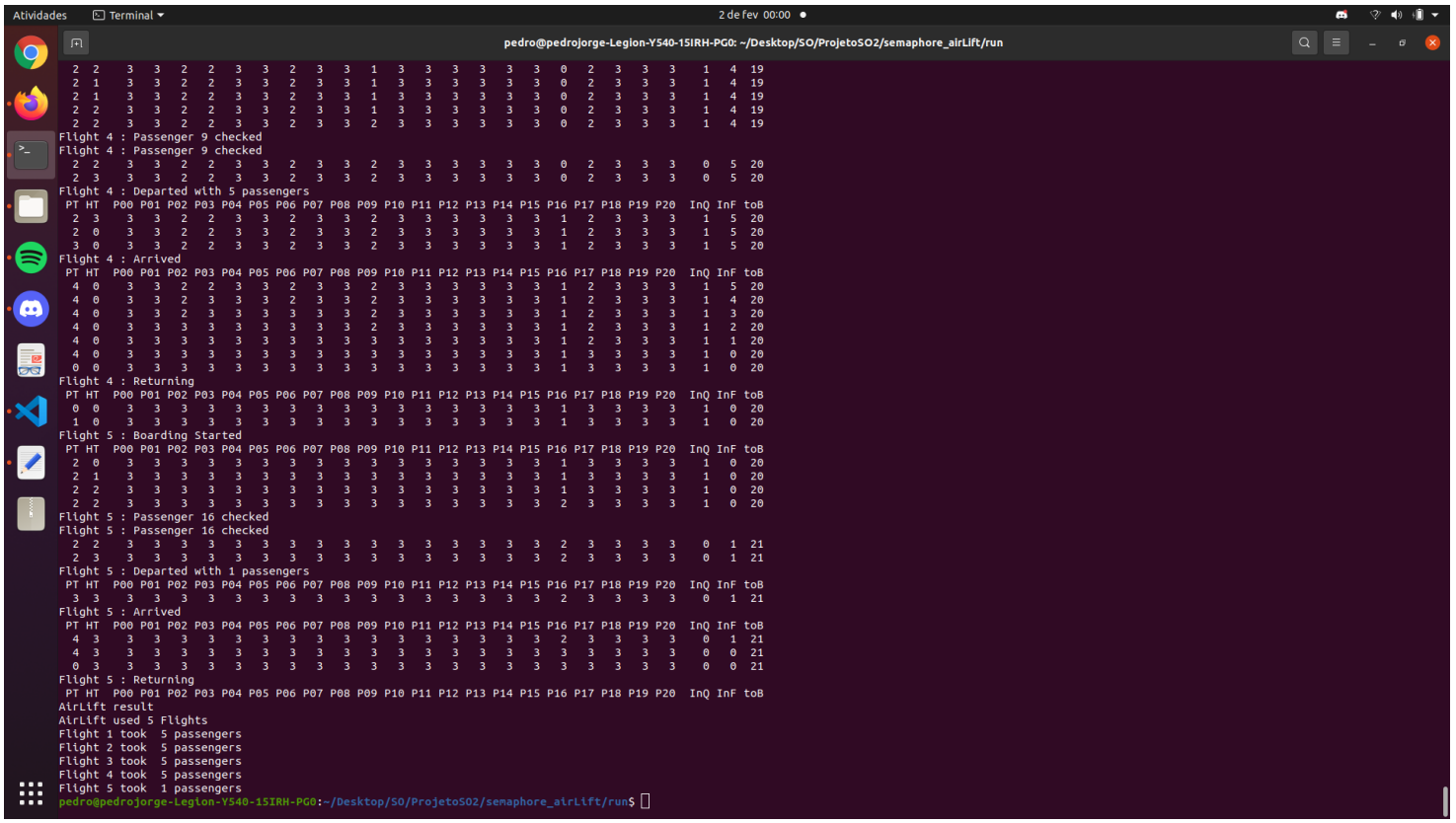



```
Flight 2 : Passenger 5 checked
  2  2    2   3   0   0   0   2   0   0   3   0   2   0   3   2   0   3   0   0   3   0   2    0   5  10
  2  3    2   3   0   0   0   2   0   0   3   0   2   0   3   2   0   3   0   0   3   0   2    0   5  10
Flight 2 : Departed with 5 passengers
 PT HT  P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20  InQ InF toB
  2  0    2   3   0   0   0   2   0   0   3   0   2   0   3   2   0   3   0   0   3   0   2    0   5  10
  3  0    2   3   0   0   0   2   0   0   3   0   2   0   3   2   0   3   0   0   3   0   2    0   5  10
  3  0    2   3   0   0   0   2   0   0   3   0   2   0   3   2   0   3   0   0   3   1   2    1   5  10
Flight 2 : Arrived
 PT HT  P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20  InQ InF toB
  4  0    2   3   0   0   0   2   0   0   3   0   2   0   3   2   0   3   0   0   3   1   2    1   5  10
  4  0    3   3   0   0   0   2   0   0   3   0   2   0   3   2   0   3   0   0   3   1   2    1   4  10
  4  0    3   3   0   0   0   2   0   0   3   0   3   0   3   2   0   3   0   0   3   1   2    1   3  10
  4  0    3   3   0   0   0   2   0   0   3   0   3   0   3   2   0   3   0   0   3   1   3    1   2  10
  4  0    3   3   0   0   0   2   0   0   3   0   3   0   3   3   0   3   0   0   3   1   3    1   1  10
  4  0    3   3   0   0   0   3   0   0   3   0   3   0   3   3   0   3   0   0   3   1   3    1   0  10
  0  0    3   3   0   0   0   3   0   0   3   0   3   0   3   3   0   3   0   0   3   1   3    1   0  10
Flight 2 : Returning
 PT HT  P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20  InQ InF toB
  0  0    3   3   0   0   0   3   0   0   3   0   3   0   3   3   0   3   0   0   3   1   3    1   0  10
  1  0    3   3   0   0   0   3   0   0   3   0   3   0   3   3   0   3   0   0   3   1   3    1   0  10
Flight 3 : Boarding Started
 PT HT  P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20  InQ InF toB
  2  0    3   3   0   0   0   3   0   0   3   0   3   0   3   3   0   3   0   0   3   1   3    1   0  10
  2  1    3   3   0   0   0   3   0   0   3   0   3   0   3   3   0   3   0   0   3   1   3    1   0  10
  2  2    3   3   0   0   0   3   0   0   3   0   3   0   3   3   0   3   0   0   3   1   3    1   0  10
  2  2    3   3   0   0   0   3   0   0   3   0   3   0   3   3   0   3   0   0   3   2   3    1   0  10
Flight 3 : Passenger 19 checked
Flight 3 : Passenger 19 checked
  2  2    3   3   0   0   0   3   0   0   3   0   3   0   3   3   0   3   0   0   3   2   3    0   1  11
  2  1    3   3   0   0   0   3   0   0   3   0   3   0   3   3   0   3   0   0   3   2   3    0   1  11
  2  1    3   3   0   0   0   3   0   0   3   0   3   0   3   3   0   3   0   0   3   2   3    0   1  11
  2  1    3   3   0   0   1   3   0   0   3   0   3   0   3   3   0   3   0   0   3   2   3    1   1  11
  2  2    3   3   0   0   1   3   0   0   3   0   3   0   3   3   0   3   0   0   3   2   3    1   1  11
  2  2    3   3   0   0   2   3   0   0   3   0   3   0   3   3   0   3   0   0   3   2   3    1   1  11
Flight 3 : Passenger 4 checked
Flight 3 : Passenger 4 checked
  2  2    3   3   0   0   2   3   0   0   3   0   3   0   3   3   0   3   0   0   3   2   3    0   2  12
  2  1    3   3   0   0   2   3   0   0   3   0   3   0   3   3   0   3   0   0   3   2   3    0   2  12
  2  1    3   3   0   0   2   3   0   0   3   0   3   0   3   3   0   3   0   0   3   2   3    0   2  12
  2  1    3   3   0   0   2   3   0   0   3   0   3   1   3   3   0   3   0   0   3   2   3    1   2  12
  2  2    3   3   0   0   2   3   0   0   3   0   3   1   3   3   0   3   0   0   3   2   3    1   2  12
  2  2    3   3   0   0   2   3   0   0   3   0   3   2   3   3   0   3   0   0   3   2   3    1   2  12
Flight 3 : Passenger 11 checked
Flight 3 : Passenger 11 checked
  2  2    3   3   0   0   2   3   0   0   3   0   3   2   3   3   0   3   0   0   3   2   3    0   3  13
  2  1    3   3   0   0   2   3   0   0   3   0   3   2   3   3   0   3   0   0   3   2   3    0   3  13
  2  1    3   3   0   0   2   3   0   0   3   0   3   2   3   3   0   3   0   0   3   2   3    0   3  13
  2  1    3   3   0   0   2   3   0   1   3   0   3   2   3   3   0   3   0   0   3   2   3    1   3  13
  2  2    3   3   0   0   2   3   0   1   3   0   3   2   3   3   0   3   0   0   3   2   3    1   3  13
  2  2    3   3   0   0   2   3   0   2   3   0   3   2   3   3   0   3   0   0   3   2   3    1   3  13
Flight 3 : Passenger 7 checked
Flight 3 : Passenger 7 checked
  2  2    3   3   0   0   2   3   0   2   3   0   3   2   3   3   0   3   0   0   3   2   3    0   4  14
  2  1    3   3   0   0   2   3   0   2   3   0   3   2   3   3   0   3   0   0   3   2   3    0   4  14
  2  1    3   3   0   0   2   3   0   2   3   0   3   2   3   3   0   3   0   0   3   2   3    0   4  14
```




```
  2  1    3   3   0   0   2   3   0   2   3   0   3   2   3   3   0   3   0   0   3   2   3    0   4  14
  2  1    3   3   0   0   2   3   0   2   3   0   3   2   3   3   0   3   0   0   3   2   3    0   4  14
  2  1    3   3   0   0   2   3   0   2   3   0   3   2   3   3   1   3   0   0   3   2   3    1   4  14
  2  2    3   3   0   0   2   3   0   2   3   0   3   2   3   3   1   3   0   0   3   2   3    1   4  14
  2  2    3   3   0   0   2   3   0   2   3   0   3   2   3   3   2   3   0   0   3   2   3    1   4  14
Flight 3 : Passenger 14 checked
Flight 3 : Passenger 14 checked
  2  2    3   3   0   0   2   3   0   2   3   0   3   2   3   3   2   3   0   0   3   2   3    0   5  15
  2  3    3   3   0   0   2   3   0   2   3   0   3   2   3   3   2   3   0   0   3   2   3    0   5  15
Flight 3 : Departed with 5 passengers
 PT HT  P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20  InQ InF toB
  2  0    3   3   0   0   2   3   0   2   3   0   3   2   3   3   2   3   0   0   3   2   3    0   5  15
  3  0    3   3   0   0   2   3   0   2   3   0   3   2   3   3   2   3   0   0   3   2   3    0   5  15
  3  0    3   3   0   1   2   3   0   2   3   0   3   2   3   3   2   3   0   0   3   2   3    1   5  15
Flight 3 : Arrived
 PT HT  P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20  InQ InF toB
  4  0    3   3   0   1   2   3   0   2   3   0   3   2   3   3   2   3   0   0   3   2   3    1   5  15
  4  0    3   3   0   1   3   3   0   2   3   0   3   2   3   3   2   3   0   0   3   2   3    1   4  15
  4  0    3   3   0   1   3   3   0   3   3   0   3   2   3   3   2   3   0   0   3   2   3    1   3  15
  4  0    3   3   0   1   3   3   0   3   3   0   3   2   3   3   3   3   0   0   3   2   3    1   2  15
  4  0    3   3   0   1   3   3   0   3   3   0   3   3   3   3   3   3   0   0   3   2   3    1   1  15
  4  0    3   3   0   1   3   3   0   3   3   0   3   3   3   3   3   3   0   0   3   3   3    1   0  15
  0  0    3   3   0   1   3   3   0   3   3   0   3   3   3   3   3   3   0   0   3   3   3    1   0  15
Flight 3 : Returning
 PT HT  P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20  InQ InF toB
  0  0    3   3   0   1   3   3   0   3   3   0   3   3   3   3   3   3   0   0   3   3   3    1   0  15
  0  0    3   3   0   1   3   3   0   3   3   0   3   3   3   3   3   3   0   1   3   3   3    2   0  15
  1  0    3   3   0   1   3   3   0   3   3   0   3   3   3   3   3   3   0   1   3   3   3    2   0  15
Flight 4 : Boarding Started
 PT HT  P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20  InQ InF toB
  2  0    3   3   0   1   3   3   0   3   3   0   3   3   3   3   3   3   0   1   3   3   3    2   0  15
  2  1    3   3   0   1   3   3   0   3   3   0   3   3   3   3   3   3   0   1   3   3   3    2   0  15
  2  2    3   3   0   1   3   3   0   3   3   0   3   3   3   3   3   3   0   1   3   3   3    2   0  15
  2  2    3   3   0   2   3   3   0   3   3   0   3   3   3   3   3   3   0   1   3   3   3    2   0  15
Flight 4 : Passenger 3 checked
Flight 4 : Passenger 3 checked
  2  2    3   3   0   2   3   3   0   3   3   0   3   3   3   3   3   3   0   1   3   3   3    1   1  16
  2  1    3   3   0   2   3   3   0   3   3   0   3   3   3   3   3   3   0   1   3   3   3    1   1  16
  2  1    3   3   0   2   3   3   0   3   3   0   3   3   3   3   3   3   0   1   3   3   3    1   1  16
  2  2    3   3   0   2   3   3   0   3   3   0   3   3   3   3   3   3   0   1   3   3   3    1   1  16
  2  2    3   3   0   2   3   3   0   3   3   0   3   3   3   3   3   3   0   2   3   3   3    1   1  16
Flight 4 : Passenger 17 checked
Flight 4 : Passenger 17 checked
  2  2    3   3   0   2   3   3   0   3   3   0   3   3   3   3   3   3   0   2   3   3   3    0   2  17
  2  1    3   3   0   2   3   3   0   3   3   0   3   3   3   3   3   3   0   2   3   3   3    0   2  17
  2  1    3   3   0   2   3   3   0   3   3   0   3   3   3   3   3   3   0   2   3   3   3    0   2  17
  2  1    3   3   0   2   3   3   1   3   3   0   3   3   3   3   3   3   0   2   3   3   3    1   2  17
  2  2    3   3   0   2   3   3   1   3   3   0   3   3   3   3   3   3   0   2   3   3   3    1   2  17
  2  2    3   3   0   2   3   3   2   3   3   0   3   3   3   3   3   3   0   2   3   3   3    1   2  17
Flight 4 : Passenger 6 checked
Flight 4 : Passenger 6 checked
  2  2    3   3   0   2   3   3   2   3   3   0   3   3   3   3   3   3   0   2   3   3   3    0   3  18
  2  1    3   3   0   2   3   3   2   3   3   0   3   3   3   3   3   3   0   2   3   3   3    0   3  18
  2  1    3   3   0   2   3   3   2   3   3   0   3   3   3   3   3   3   0   2   3   3   3    0   3  18
  2  1    3   3   1   2   3   3   2   3   3   0   3   3   3   3   3   3   0   2   3   3   3    1   3  18
  2  2    3   3   1   2   3   3   2   3   3   0   3   3   3   3   3   3   0   2   3   3   3    1   3  18
```


Atividades
Terminal
1 de fev 23:59
pedro@pedrojorge-Legion-Y540-15IRH-PG0: ~/Desktop/SO/ProjetoSO2/semaphore_airLift/run
Flight 4 : Passenger 2 checked
Flight 4 : Passenger 2 checked
Flight 4 : Passenger 9 checked
Flight 4 : Passenger 9 checked
Flight 4 : Departed with 5 passengers
PT HT P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20 InQ InF toB
Flight 4 : Arrived
PT HT P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20 InQ InF toB
Flight 4 : Returning
PT HT P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20 InQ InF toB
Flight 5 : Boarding Started
PT HT P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20 InQ InF toB
Flight 5 : Passenger 16 checked
Flight 5 : Passenger 16 checked
Flight 5 : Departed with 1 passengers
PT HT P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20 InQ InF toB
Flight 5 : Arrived
PT HT P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20 InQ InF toB
Flight 5 : Returning
PT HT P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20 InQ InF toB
AirLift result
AirLift used 5 Flights
Flight 1 took 5 passengers


Atividades
Terminal
2 de fev 00:00
pedro@pedrojorge-Legion-Y540-15IRH-PG0: ~/Desktop/SO/ProjetoSO2/semaphore_airLift/run
Flight 4 : Passenger 9 checked
Flight 4 : Passenger 9 checked
Flight 4 : Departed with 5 passengers
PT HT P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20 InQ InF toB
Flight 4 : Arrived
PT HT P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20 InQ InF toB
Flight 4 : Returning
PT HT P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20 InQ InF toB
Flight 5 : Boarding Started
PT HT P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20 InQ InF toB
Flight 5 : Passenger 16 checked
Flight 5 : Passenger 16 checked
Flight 5 : Departed with 1 passengers
PT HT P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20 InQ InF toB
Flight 5 : Arrived
PT HT P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20 InQ InF toB
Flight 5 : Returning
PT HT P00 P01 P02 P03 P04 P05 P06 P07 P08 P09 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20 InQ InF toB
AirLift result
AirLift used 5 Flights
Flight 1 took 5 passengers
Flight 2 took 5 passengers
Flight 3 took 5 passengers
Flight 4 took 5 passengers
Flight 5 took 1 passengers
pedro@pedrojorge-Legion-Y540-15IRH-PG0:~/Desktop/SO/ProjetoSO2/semaphore_airLift/run$

# Conclusão

Concluindo, a realização deste projeto aumento o meu conhecimento acerca de semáforos e da sua importância na sincronização de processos. De maneira geral, e do meu ponto de vista, o trabalho foi bem conseguido visto que é possível fazer vários voos sem a existência de deadlocks e as variáveis do tipo FULL_STAT presentes no output vão de acordo com os acontecimentos que vão decorrendo.

# Bibliografia

De maneira a auxiliar-me neste projeto recorri aos seguintes conteúdos:

- Slides Aulas Teóricas;
- Guiões Práticos;
- Fóruns do StackOverFlow;
- https://www.geeksforgeeks.org/semaphores-in-process-synchronization/ ;
- https://www.tutorialspoint.com/semaphores-in-operating-system .