UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS SOBRAL

ENGENHARIA DA COMPUTAÇÃO

SISTEMAS OPERACIONAIS

DENILSON GOMES VAZ DA SILVA 374872

MELISSE PONTES CABRAL 406643

THREADS, CONCORRÊNCIA E CONTROLE DE SEÇÃO CRÍTICA

SOBRAL

2018

1. INTRODUÇÃO

Muitas às vezes, a execução do programa principal não precisa, nem deve, esperar pelo fim da execução de uma função chamada. Assim, enquanto uma função executa suas instruções o programa principal pode prosseguir com a sua execução [1].

Cada segmento de fluxo independente do programa principal é chamado de Thread e pode executar de forma concorrente com a execução do programa principal e de outras Threads.

A execução concorrente é muitas às vezes confundida com a execução paralela, no entanto não são a mesma coisa. A programação paralela ocorre de forma simultânea em processadores diferentes, enquanto em um processador temos um processo temos outro processador para outro processo. Na programação concorrente, os processos concorrem ao mesmo tempo para fazer o uso de um mesmo processador.

Com o uso de técnicas de multiprogramação é possível simular uma execução paralela em um único processador, obtendo na verdade uma execução concorrente.

Com várias Threads em execução por vez é possível que duas ou mais Threads pretendam acessar a uma mesma região e comprometam a operação uma a da outra. Uma região dessa é nomeada de região crítica.

Dessa forma, devemos garantir que apenas uma Thread acesse a região crítica por vez, para isso existem vários mecanismos, entre eles o semáforo. O semáforo funciona como uma sinalização de disponibilidade da região crítica, e forma uma lista de espera para acesso a essa região.

Com um semáforo é possível executar às Threads de forma concorrente com a garantia de que uma Thread não vai interferir na operação da outra.

2. DESCRIÇÃO DO PROBLEMA

O problema consiste em gerenciar um aeroporto com 2 (duas) pistas. Neste caso, em cada pista pode ter apenas um avião realizando o procedimento de decolagem por vez. Dois aviões podem decolar ao mesmo tempo, contanto que usem pistas diferentes, enquanto um avião estiver no processo de decolagem em uma pista outros aviões não podem usar essa pista.

O processo de decolagem consiste em outras 6 (seis) tarefas. Estas 6 (seis) tarefas e suas devidas durações são:

- manobra: 3 a 7 segundos
- taxiar: 2 a 5 segundos
- posicionar-se na cabeceira: 1 a 4 segundos
- acelerar: 3 a 5 segundos
- decolar: 4 a 7 segundos
- afastar: 2 a 6 segundos

Neste caso, temos que efetuar a decolagem de 6 aviões. O controle do acesso à pista deve ser feito de forma a garantir que cada pista seja usada por apenas um avião por vez.

3. IMPLEMENTAÇÃO DA SOLUÇÃO

A implementação foi feita utilizando a linguagem Java, através de ferramentas da própria linguagem fomos capaz de utilizar a concorrência de forma simples. Foram utilizadas algumas bibliotecas disponíveis para a linguagem, dentre os principais recursos utilizados estão: *ExecutorService, Executors, Semaphore, TimeUnit, SecureRandom*.

A através do *ExecutorService* foi possível executar tarefas da classe *Aviao* a qual implementa a interface *Runnable* o que tornou possível sua execução independente. A classe *Aviao* era responsável pelos procedimentos de decolagem que foram implementados para serem executados no no método *run().*

Os métodos responsáveis pelo processo de decolagem de *Aviao* fazem usos dos seguintes recursos *Semaphore, TimeUnit, SecureRandom.* O primeiro recurso

utilizado *Semaphore* faz o controle de acesso as seções críticas do programa e os recursos *TimeUnit* e *SecureRandom* são utilizados com o propósito de simular o tempo de ocupação da seção crítica.

A classe principal do programa é a classe *Aeroporto*, encarregada pelo *main()* que possui as instruções de execução do programa e a inicialização dos recursos a serem disputado.

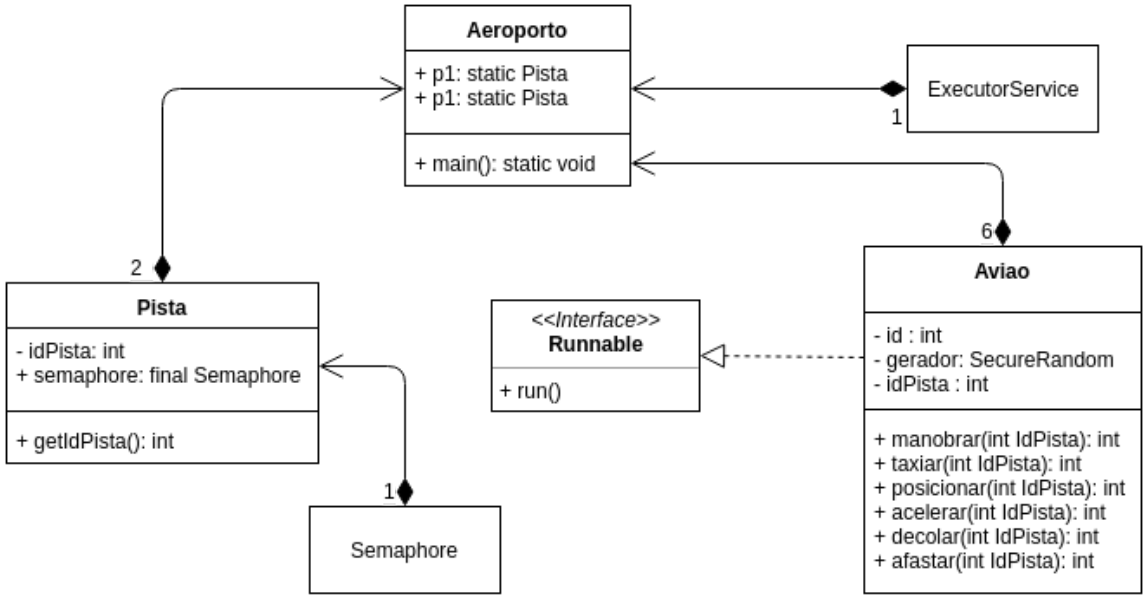

Fig. 1 Modelagem da implementação

A execução do programa dessa maneira inicia-se em *Aeroporto* onde são inicializados as duas seções críticas, junto a cada seção crítica é criado um *Semaphore responsável pelo controle de acesso* a ela. A partir da criação dessas seções o programa segue criando os objetos da classe *Aviao* e dando início ao processo de decolagem, é nesse momento que a execução de cada um deles é feito pelo *ExecutorService* que inicia para cada *Aviao* uma Thread. Dessa a concorrência do programa existe e segue de acordo com a disponibilidade das seções críticas e respeitando a ordem de chegada de cada processo na disputa dos recursos.

4. RESULTADOS

Com a execução do programa foi observado que obtemos o comportamento esperado. A concorrência pode ser observadas nas saidas das execuções abaixo.

Execução 1:

```
AV0 Iniciando processo de decolagem. Na Pista 0
AV2 Iniciando processo de decolagem. Na Pista 1
AV2 Manobrando... 6520ms
AV0 Manobrando... 4754ms
AV0 Taxiando... 3391ms
AV2 Taxiando... 3382ms
AV0 Posicionando...2971ms
AV0 Acelerando.. 4902ms
AV2 Posicionando...1466ms
AV2 Acelerando.. 4470ms
AV2 Decolando...5689ms
AV0 Decolando...4539ms
AV0 Afastando... Na Pista 0 Tempo: 5960ms
AV2 Afastando... Na Pista 1 Tempo: 5419ms
AV1 Iniciando processo de decolagem. Na Pista 0
AV1 Manobrando... 5922ms
AV4 Iniciando processo de decolagem. Na Pista 1
AV4 Manobrando... 4150ms
AV4 Taxiando... 4109ms
```

AV1 Taxiando... 3791ms
AV4 Posicionando...1396ms
AV4 Acelerando.. 4457ms
AV1 Posicionando...2674ms
AV1 Acelerando.. 4550ms
AV4 Decolando...5806ms
AV1 Decolando...6245ms
AV4 Afastando... Na Pista 1 Tempo: 3368ms
AV1 Afastando... Na Pista 0 Tempo: 4205ms
AV5 Iniciando processo de decolagem. Na Pista 1
AV5 Manobrando... 5670ms
AV3 Iniciando processo de decolagem. Na Pista 0
AV3 Manobrando... 4225ms
AV5 Taxiando... 2880ms
AV3 Taxiando... 2923ms
AV5 Posicionando...2827ms
AV5 Acelerando.. 4892ms
AV3 Posicionando...2523ms
AV3 Acelerando.. 3875ms
AV5 Decolando...5225ms
AV3 Decolando...4410ms
AV5 Afastando... Na Pista 1 Tempo: 2191ms
AV3 Afastando... Na Pista 0 Tempo: 4537ms

Execução 2:

AV0 Iniciando processo de decolagem. Na Pista 1
AV4 Iniciando processo de decolagem. Na Pista 0
AV0 Manobrando... 3120ms
AV4 Manobrando... 5526ms
AV0 Taxiando... 2696ms
AV4 Taxiando... 3080ms
AV0 Posicionando...3193ms
AV0 Acelerando.. 3460ms
AV4 Posicionando...1190ms
AV4 Acelerando.. 4456ms
AV0 Decolando...5576ms
AV4 Decolando...6849ms

AV0 Afastando... Na Pista 1 Tempo: 3820ms
AV4 Afastando... Na Pista 0 Tempo: 4560ms
AV1 Iniciando processo de decolagem. Na Pista 1
AV1 Manobrando... 4899ms
AV5 Iniciando processo de decolagem. Na Pista 0
AV5 Manobrando... 5914ms
AV1 Taxiando... 4371ms
AV1 Posicionando...2347ms
AV1 Acelerando.. 3217ms
AV5 Taxiando... 2275ms
AV1 Decolando...5666ms
AV5 Posicionando...3456ms
AV5 Acelerando.. 4490ms
AV5 Decolando...6837ms
AV1 Afastando... Na Pista 1 Tempo: 5374ms
AV2 Iniciando processo de decolagem. Na Pista 1
AV2 Manobrando... 4995ms
AV5 Afastando... Na Pista 0 Tempo: 4946ms
AV2 Taxiando... 3867ms
AV2 Posicionando...2543ms
AV2 Acelerando.. 4998ms
AV2 Decolando...6958ms
AV2 Afastando... Na Pista 1 Tempo: 2026ms
AV3 Iniciando processo de decolagem. Na Pista 1
AV3 Manobrando... 5743ms
AV3 Taxiando... 2972ms
AV3 Posicionando...2438ms
AV3 Acelerando.. 4770ms
AV3 Decolando...4526ms
AV3 Afastando... Na Pista 1 Tempo: 3195ms

Execução 3:

AV0 Iniciando processo de decolagem. Na Pista 1
AV1 Iniciando processo de decolagem. Na Pista 0
AV0 Manobrando... 3512ms
AV1 Manobrando... 5610ms
AV0 Taxiando... 2817ms

AV1 Taxiando... 2357ms
AV0 Posicionando...1743ms
AV0 Acelerando.. 3402ms
AV0 Decolando...4480ms
AV1 Posicionando...3830ms
AV1 Acelerando.. 4670ms
AV0 Afastando... Na Pista 1 Tempo: 2224ms
AV1 Decolando...5542ms
AV5 Iniciando processo de decolagem. Na Pista 1
AV5 Manobrando... 6829ms
AV1 Afastando... Na Pista 0 Tempo: 4213ms
AV2 Iniciando processo de decolagem. Na Pista 0
AV2 Manobrando... 6204ms
AV5 Taxiando... 2650ms
AV5 Posicionando...2093ms
AV5 Acelerando.. 4204ms
AV5 Decolando...5023ms
AV2 Taxiando... 3715ms
AV2 Posicionando...1743ms
AV2 Acelerando.. 4268ms
AV5 Afastando... Na Pista 1 Tempo: 4630ms
AV2 Decolando...5841ms
AV2 Afastando... Na Pista 0 Tempo: 3625ms
AV3 Iniciando processo de decolagem. Na Pista 0
AV3 Manobrando... 4662ms
AV3 Taxiando... 3913ms
AV3 Posicionando...1349ms
AV3 Acelerando.. 3408ms
AV3 Decolando...5225ms
AV3 Afastando... Na Pista 0 Tempo: 5716ms
AV4 Iniciando processo de decolagem. Na Pista 0
AV4 Manobrando... 4597ms
AV4 Taxiando... 2955ms
AV4 Posicionando...2882ms
AV4 Acelerando.. 3693ms
AV4 Decolando...5552ms
AV4 Afastando... Na Pista 0 Tempo: 4991ms

5. CONCLUSÃO

Com a realização deste trabalho conseguimos realizar a execução de 6 (seis) tarefas independentes do programa principal de forma concorrente e fazer o controle de 2 (duas) regiões crítica, neste caso as duas pistas de decolagem.

Podemos observar que para execuções diferentes, temos diferentes ordens de decolagem dos aviões, o que significa que obtemos uma aleatoriedade na execução das Threads responsáveis pela decolagem dos aviões. Além de controlar as pistas para que apenas um avião use uma determinada pista por vez, ou seja, quando um avião começa o processo de decolagem em uma pista, esta pista permanece bloqueada até que o avião termine o processo de decolagem.

REFERÊNCIAS

FABIO JORDÃO .O que são threads em um processador? (2011). Disponível em: <https://www.tecmundo.com.br/internet/9669-o-que-sao-threads-em-um-processador-.htm>

Andrei Rimsa Álvares .Programação Concorrente. Disponível em: <http://homepages.dcc.ufmg.br/~rimsa/documents/decom009/lessons/Aula09.pdf>