Universidad Nacional Autónoma de México

Facultad de ingeniería

Sistemas operativos

Proyecto 2 de Sistemas Operativos

Un día en el aeropuerto Felipe Ángeles

Profesor: Wolf Iszaevich Gunnar Eyal

Alumnos: Leyva Mercado Christian Alejandro y Velasco Pachuca Bryan

Grupo: 06

Situación

Se acaba de inaugurar el nuevo aeropuerto Felipe Ángeles en el que se cuentan con las siguientes zonas de operación:

- Pista de aterrizaje/despegue.
- Torre de control.
- Zona de espera(terrestre).
- Zona de espera (aérea).
- Zona de carga/descarga de pasajeros.
- Zona de descarga/carga de mercancía.

Se sabe que se tiene un flujo continuo de peticiones de la pista de aterrizaje las cuales son gestionadas por la torre de control, la torre de control deberá administrar los aterrizajes y despegues según como lleguen las peticiones de los aviones (sin generar inanición a alguno de los aviones), cada uno de los aviones que aterriza volverá a despegar tras dejar a sus pasajeros o mercancía y ser llenado nuevamente con los pasajeros o la mercancía.

El andén de carga/descarga de pasajeros permite la operación de hasta 10 aviones comerciales al mismo tiempo y el andén de carga/descarga de mercancía permite la operación de hasta 6 aviones de carga también al mismo tiempo.

Cuando un avión no puede aterrizar se deja volando en la zona de espera aérea y cuando un avión no puede despegar se deja esperando en la zona de espera terrestre.

Consecuencias nocivas de la concurrencia:

En un aeropuerto la peor situación de concurrencia se da cuando existe un excedente de tráfico aéreo que obliga a los aviones a sobrevolar el aeropuerto o retrasar el despegue más de lo debido, incurriendo en retrasos o inclusive accidentes. Por esta razón una correcta administración de los espacios es primordial para cualquier torre de control.

Eventos concurrentes para los cuales el ordenamiento no resulte importante:

Dado que el tiempo de permanencia de cada avión en el aeropuerto es una característica intrínseca de cada cargamento, el tiempo que estos tarden en despegar de nuevo generará un nuevo ordenamiento en los despegues, por ello su orden de salida no resulta relevante.

Documentación

Disclaimer

En el proyecto existen 2 archivos de Python, la versión *aeropuertoBackEnd.py* que muestra la solución en línea de comandos y la versión *Grafico.py* el cual muestra de manera más atractiva los resultados, no obstante, se nos presentaron problemas para manipular los gráficos de Python que tras no resolverlos incurrió en su incompletitud.

Por el motivo anteriormente expuesto, se dará un mayor hincapié en la solución en línea de comando (*aeropuertoBackEnd.py*).

Mecanismos de sincronización empleados

Los mecanismos de sincronización utilizados son los siguientes:

- **Rendezvous**: El rendezvous fue utilizado para establecer una comunicación efectiva entre el avión y la torre de control y de esta manera asignarle la pista al avión.

  Declaración:

```
mutexRadioAvion = Semaphore(1)
mensaje = 0
mutexRadioTorreDeControl = Semaphore(0)
```

  Implementación:

  En método *comunicacion()*:

```
def comunicacion(ref:int):
    global mutexRadioAvion, mutexRadioTorreDeControl, mensaje
    mutexRadioAvion.acquire()
    mensaje = ref
    mutexRadioTorreDeControl.release()
```

  En método *torreDeControl()*:

```
def torreDeControl():
    global pistaDeAviacion, mutexRadioTorreDeControl,mutexRadioAvion, mensaje
    print("**Aquí torre de control, iniciamos operaciones.**")
    while True:
        mutexRadioTorreDeControl.acquire()
        print("**Aquí torre de control, avion No. " + str(mensaje) + " puede
        pistaDeAviacion.release()
        mutexRadioAvion.release()
```

- **Mutex:** El mutex fue utilizado para proteger la pista de aviación y que esta no sea utilizada por más de un avión, evitando colisiones.

Declaración:

```
pistaDeAviacion = Semaphore(0)
```

Implementación:

La implementación se realizó prácticamente en todo el código y para no confundir al lector se mostrará su implementación en la descripción algorítmica.

- **Multiplex:** El multiplex fue utilizado para determinar la cantidad de aviones que podían utilizar los andenes de pasajeros o mercancía.

  Declaración:

```
andenPasajeros = Semaphore(10)
andenMercancia = Semaphore(6)
```

  Implementación:

  En método *descarga()*:

```
def descargaPAX(ref:int):
    global pistaDeAviacion,andenPasajeros
    andenPasajeros.acquire()
    sleep(random.randrange(2,5))
    print("\tAvión comercial No. " + str(ref
```

  En método *carga()*:

```
def cargaPAX(ref:int):
    global andenPasajeros
    sleep(random.randrange(2,5))
    andenPasajeros.release()
    print("\tAvión comercial No. "
```

Lógica de operación

Identificación del estado compartido:

Para la solución a este problema se presentan varios estados compartidos, siendo el más importante el que se encuentra entre los métodos de *comunicación()* y *torreDeControl(),* pues es gracias a estos que es posible la asignación de la pista.

En el método de comunicación se declara que los identificadores mutexRadioAvion, mutexRadioTorreDeControl y mensaje pertenecen a un contexto global por lo que su utilización es compartida.

```
def comunicacion(ref:int):
    global mutexRadioAvion, mutexRadioTorreDeControl, mensaje
    mutexRadioAvion.acquire()
    mensaje = ref
    mutexRadioTorreDeControl.release()
```

En el método se puede observar como para establecer la comunicación con la torre de control es necesario manipular el mensaje que se desea enviar (el cual contiene el número de avión), para lograr esto se utiliza un mutex que protege la integridad del mensaje.

Tras guardar el mensaje se avisa a la torre de control que se espera una respuesta, lo cual activa la secuencia de respuesta en el método *torreDeControl()*.

```
def torreDeControl():
    global pistaDeAviacion, mutexRadioTorreDeControl,mutexRadioAvion, mensaje
    print("** Aquí torre de control, iniciamos operaciones.**")
    while True:
        mutexRadioTorreDeControl.acquire()
        print("**  Aquí torre de control, avion No. {} puede ocupar la pist
        pistaDeAviacion.release()
        mutexRadioAvion.release()
```

Recibido el mensaje en torre de control, se procede a reservar el recurso de la pista para el avión que la haya pedido.

```
def avionComercial(ref:int):
    global pistaDeAviacion
    start = time()
    print("\t  Aquí avión comercial No. " + str(ref) +
    comunicacion(ref)
    pistaDeAviacion.acquire()
    print("\tAvión comercial " + str(ref) + " en tierra.")
```

Descripción algorítmica del avance de cada hilo/proceso:

El avance y creación de los hilos se origina en 2 fuentes, la primera es el método *traficoAereo()* el cual se encarga de crear de manera aleatoria los aviones que desean llegar al aeropuerto, pudiendo ser comerciales o cargueros.

```
def traficoAereo():
    ref = 1
    while True:
        opcion = random.randint(0, 1)
        if opcion == 0 :
            Thread(target = avionComercial, args= [ref]).start()
            ref += 1
            sleep(random.randrange(1,3))
        else:
            Thread(target = avionCarguero, args = [ref]).start()
            ref +=1
            sleep(random.randrange(2,5))
```

En dicho método se inicia el flujo de proceso de cada avión.

Ambos aviones tienen un proceso similar de ejecución, en donde primero se establece una comunicación con la torre de control para reservar el uso de la pista, posteriormente y tras aterrizar, se realizará la descarga de su contenido para lo cual requiere reservar un espacio en su respectivo andén (de pasajeros o de mercancía). Finalizada la descarga, el avión procede a cargarse de nuevo y después establecer comunicación con la torre de control para reservar la pista y poder despegar.

Método de ***avionComercial()***

```
def avionComercial(ref:int):
    global pistaDeAviacion
    start = time()
    print("\t ✈ Aquí avión comercial No. " + str(ref) + ", solic
    comunicacion(ref)
    pistaDeAviacion.acquire()
    print("\tAvión comercial " + str(ref) + " en tierra.")
    descargaPAX(ref)
    cargaPAX(ref)
    #Avión en zona de espera terrestre
    comunicacion(ref)
    pistaDeAviacion.acquire()
    finish = time() - start
    print(">>Avión comercial No. " + str(ref) + " en aire, {0:.1f}
```

Método que simula la descarga de pasajeros tras adquirir un espacio en su respectivo andén.

```
def descargaPAX(ref:int):
    global pistaDeAviacion,andenPasajeros
    andenPasajeros.acquire()
    sleep(random.randrange(2,5))
    print("\tAvión comercial No. " + str(ref) + " desembarcó
```

Método que simula la carga de pasajeros tras adquirir un espacio en su respectivo andén.

```
def cargaPAX(ref:int):
    global andenPasajeros
    sleep(random.randrange(2,5))
    andenPasajeros.release()
    print("\tAvión comercial No. " + str(ref) + " tripulado,
```

La operatividad de los aviones es la misma para los aviones comerciales y cargueros.

Descripción del entorno de desarrollo

¿Qué lenguaje emplean? ¿Qué versión?

El lenguaje utilizado es Python en su versión 3.9.10

¿Qué bibliotecas más allá del estándar del lenguaje?

Para el desarrollo de la solución se emplearon las siguientes bibliotecas:

1. **threading:** Esta biblioteca es la más importante del proyecto pues nos permitió manipular diferentes hilos y correrlos cuando fuese necesario, además de proporcionar la estructura de semáforos.
   ```
   from threading import Semaphore, Thread
   ```
2. **time:** Con esta biblioteca fue posible manipular el tiempo de cada hilo, durmiéndolos cuando fuese necesario y obteniendo el tiempo actual del programa para calcular el desempeño de cada avión/hilo.
   ```
   from time import sleep, time
   ```
3. **random:** Con la biblioteca random se generaron números aleatorios para simular el tiempo de operación de cada avión.
   ```
   import random
   ```
4. **tkinter:** Finalmente, para la interfaz de usuario que se desarrolló se implementó tkinter la cual proporciona un conjunto de elementos gráficos que nos permitieron darle una mejor vista a la aplicación.

```
from tkinter import *
```

Por desgracia, comprobamos que a pesar de que la estructura lógica sea la correcta, tuvimos problemas para que los hilos fuesen capaces de trabajar con el frame.

¿Bajo qué sistema operativo / distribución lo desarrollaron y/o probaron?

La solución fue desarrollada y probada en el sistema operativo Windows en sus distribuciones 10 y 11.

Ejemplos o pantallazos de una ejecución exitosa

Versión gráfica:

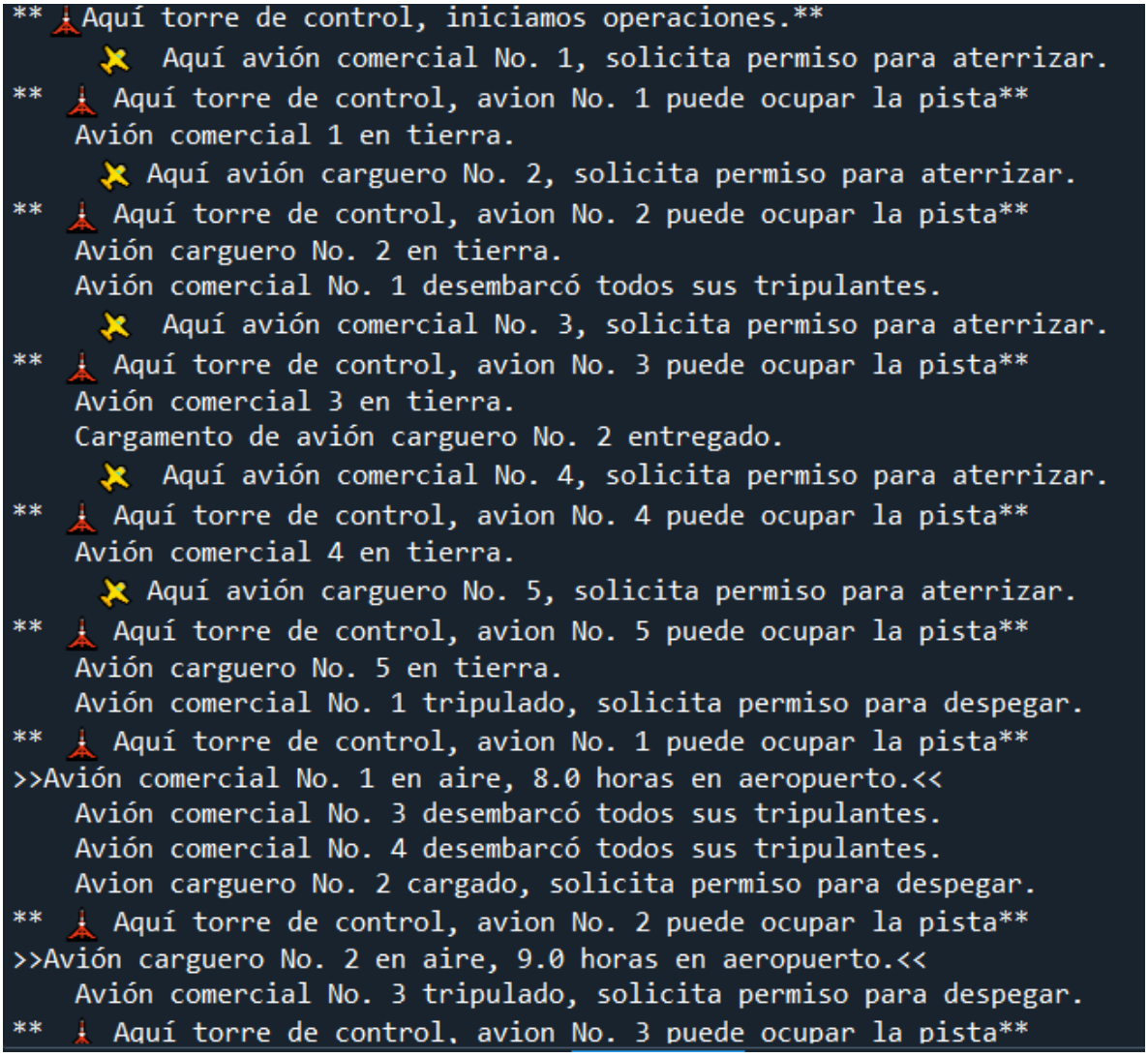
El propósito de la versión gráfica es que cada tipo de mensaje se mostrase en su área correspondiente, para poder llevar un mejor registro en la operación de cada hilo.

Los inputs son los tiempos de espera aproximados que se espería de cada tipo de avión.

Versión en línea de comando:

En una ejecución normal con un andén de pasajeros con capacidad de 10 espacios se espera que el tiempo mínimo de estancia de un avión comercial sea de 4 horas y el máximo de 10.

Para una ejecución normal con un andén de mercancía con capacidad de 6 espacios se espera que el tiempo mínimo de estancia sea de 8 horas y el máximo de 14.



Si la capacidad de los andenes se reduce a 2 para los vuelos comerciales y a uno para los vuelos cargueros:

```
andenPasajeros = Semaphore(2)
andenMercancia = Semaphore(1)
```

Entonces se podrá observar que el aeropuerto se encuentra saturando y un avión puede llegar a demorar hasta 71 horas en despegar.

```
>>Avión carguero No. 11 en aire, 35.0 horas en aeropuerto.<<
    Cargamento de avión carguero No. 12 entregado.
    Avion carguero No. 12 cargado, solicita permiso para despegar.
**  Aquí torre de control, avion No. 12 puede ocupar la pista**
>>Avión carguero No. 12 en aire, 42.0 horas en aeropuerto.<<
    Cargamento de avión carguero No. 13 entregado.
    Avion carguero No. 13 cargado, solicita permiso para despegar.
**  Aquí torre de control, avion No. 13 puede ocupar la pista**
>>Avión carguero No. 13 en aire, 49.0 horas en aeropuerto.<<
    Cargamento de avión carguero No. 16 entregado.
    Avion carguero No. 16 cargado, solicita permiso para despegar.
**  Aquí torre de control, avion No. 16 puede ocupar la pista**
>>Avión carguero No. 16 en aire, 51.0 horas en aeropuerto.<<
    Cargamento de avión carguero No. 17 entregado.
    Avion carguero No. 17 cargado, solicita permiso para despegar.
**  Aquí torre de control, avion No. 17 puede ocupar la pista**
>>Avión carguero No. 17 en aire, 60.0 horas en aeropuerto.<<
    Cargamento de avión carguero No. 18 entregado.
    Avion carguero No. 18 cargado, solicita permiso para despegar.
**  Aquí torre de control, avion No. 18 puede ocupar la pista**
>>Avión carguero No. 18 en aire, 65.0 horas en aeropuerto.<<
    Cargamento de avión carguero No. 19 entregado.
    Avion carguero No. 19 cargado, solicita permiso para despegar.
**  Aquí torre de control, avion No. 19 puede ocupar la pista**
>>Avión carguero No. 19 en aire, 71.0 horas en aeropuerto.<<
```

Conclusiones:

El desarrollo de este proyecto nos permitió desarrollar en gran medida nuestras capacidades de abstracción y estar más familiarizados con la programación paralela, acostumbrándonos a implementar mecanismos de sincronización que nos permitan generar un código de mejor calidad.

Así mismo, desarrollar una respuesta a un problema cuyo contexto está en boga también contribuyó a mejorar nuestro entendimiento en la operatividad de un aeropuerto.