Facultad de Ingeniería-UNAM

Nombre de la materia
Sistemas Operativos

Semestre: 2023-2
Prof: ING. Gunnar Eyal Wolf Iszaevich

Título del Trabajo.
Una situación cotidiana paralelizable

Número del trabajo.
Proyecto 2
Nombre del estudiante.

Miranda Barajas Victor
Grupo
06

Fecha de entrega.
06/Noviembre/ 2023

Planteamiento del problema:

El aeropuerto dispone de un número limitado de pistas de aterrizaje y despegue (en este caso tres), y un conjunto de aviones (diez en el ejemplo) que necesitan usar estas pistas. Los aviones pueden tener diferentes niveles de prioridad que determinan el orden en que deben ser atendidos por la torre de control para el uso de las pistas.

Características del problema:

- Concurrencia: Múltiples aviones pueden intentar acceder a las pistas al mismo tiempo.

- Recursos: Las pistas son limitadas y compartidas, por lo que estas sólo pueden ser ocupadas por un avión a la vez.

- Sincronización: Se necesita una sincronización para el acceso a las pistas de aterrizaje para que no haya conflictos.

- Priorización: En ciertas ocasiones los aviones pueden tener prioridad debido a una urgencia por aterrizar.

Modelaje:

- Gestión de pistas: Se deben asignar y liberar pistas para los aviones de manera eficiente.

- Torre de control: Planifica, gestiona y ordena el uso de las pistas.

Eventos a controlar:

- Asignación de pistas: Asegurarse de que se asignen las pistas a los aviones adecuados según su prioridad y la disponibilidad,

- Liberación de pistas: Coordinar la liberación de pistas de manera oportuna tras el uso.

Descripción de los Mecanismos de Sincronización Empleados:

- Semáforos: Controlan el acceso a las pistas de aterrizaje/despegue del aeropuerto, asegurando que solo un avión puede usar cada pista a la vez. Cada semáforo funciona como un contador que, si es mayor que cero, permite la entrada, reduciendo el contador en uno. Si es cero, el hilo que intenta entrar se bloquea hasta que otro hilo libera el recurso.

- Cola de Prioridades: Administra los aviones que esperan su turno para asignar una pista en función de su prioridad. Los elementos en esta cola se ordenan por su valor de prioridad, permitiendo que aviones con mayor urgencia sean atendidos primero.

- Variables de Condición: Proporcionan un mecanismo para que los hilos se suspendan esperando una condición y luego se reanuden cuando la condición se cumple. En este caso, se usan para hacer que los aviones esperen hasta que una pista esté disponible sin realizar una encuesta activa.

Lógica de Operación:

- Los aviones se generan con una prioridad aleatoria y se colocan en la cola de la torre de control.

- Cuando un avión intenta usar una pista, primero se verifica si hay una disponible.

- Si una pista está disponible, el avión procede; si no, espera hasta que se le notifique que una pista se ha liberado.

- Una vez que un avión termina en la pista, libera la pista y notifica a la torre de control y a otros aviones que pueden estar esperando.

Descripción Algorítmica del Avance de Cada Hilo:

Colocación en la Cola:

- Cada hilo se inicia “start()” y se coloca en la cola de prioridades de la torre de control.

- La inserción en la cola se realiza en función de la prioridad del avión, donde un número más bajo indica mayor prioridad.

Espera por Prioridad:

- El avión espera en la cola de la torre de control hasta que sea su turno de ser procesado.

- Esto se realiza mediante el método “get()” de la cola de prioridades que bloquea el hilo hasta que el avión alcanza el frente de la cola según su prioridad.

Asignación de Pistas:

- Una vez que es el turno del avión, intenta asignarse a sí mismo una de las pistas disponibles.

- El método “asigna_pista()” revisa cada una de las pistas y trata de adquirir un semáforo “Semaphore” sin bloquear “blocking=False”.

- Si una pista está disponible, el avión se asigna a esa pista.

Pista Disponible:

- Si ninguna pista está disponible, el avión debe esperar.

- El hilo entra en un estado de espera con la ayuda de una variable de condición “Condition”, liberando el bloqueo temporalmente y durmiendo hasta que se notifique que un pista se ha liberado.

Pista Asignada:

- Una vez asignada la pista, el avión simula el aterrizaje o despegue representado por una pausa en la ejecución con “time.sleep()”.

- La duración de esta operación es aleatoria, emulado variaciones reales en el tiempo que un avión necesitaría.

Liberación de la Pista:

- Después de completar su actividad en la pista, el avión libera el semáforo asociado a la pista que estaba utilizando.

- Al liberar el semáforo, otro avión que estuviera esperando puede ser notificado para proceder a usar la pista.

Notificación a la Torre de Control:

- El avión notifica a al torre de control que ha terminado su proceso llamando a “task_done()” en la cola de prioridades.

- Esto mantiene el conteo en estructuras de sincronización.

Descripción de la Interacción entre Ellos:

- Interacción a través de Semáforos: Cuando un avión adquiere un semáforo, otros aviones no pueden adquirirlo hasta que se libere, asegurando el acceso exclusivo a la pista.

- Interacción a través de la Cola de Prioridades: Los aviones interactúan indirectamente a través de la cola de la torre de control, donde la prioridad determina su orden de servicio.

- Interacción a través de Variables de Condición: Los aviones en espera son notificados a través de la variable de condición cuando una pista se libera, permitiendo que el siguiente en la cola intente tomar la pista liberada.

Código Comentado:

```
# Miranda Barajas Victor
import threading
import time
import random
import queue
import logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -
%(message)s')

class Aeropuerto:
    def __init__(self):
        self.pistas = [threading.Semaphore(1) for _ in range(3)]  # Tres
pistas disponibles
        self.torre_control = queue.PriorityQueue()  # Torre de control
gestiona prioridades
        self.espera_pistas = threading.Condition()  # Condición para esperar
pistas disponibles

    def asignar_pista(self, avion):
        asignada = False
        while not asignada:
            with self.espera_pistas:
                for i, pista in enumerate(self.pistas):
                    if pista.acquire(blocking=False):
                        logging.info(f"Avión {avion.avion_id} asignado a la
pista {i}.")
                        asignada = True
                        avion.pista_asignada = i
                        break
                if not asignada:
                    logging.info(f"Avión {avion.avion_id} esperando por una
pista.")
                    self.espera_pistas.wait()  # Espera hasta que una pista se
libere

    def liberar_pista(self, avion_id, pista_asignada):
        self.pistas[pista_asignada].release()
        with self.espera_pistas:
            logging.info(f"Avión {avion_id} liberó la pista
{pista_asignada}.")
            self.espera_pistas.notify()  # Notifica a los aviones en espera
que una pista se ha liberado

class Avion(threading.Thread):
    def __init__(self, avion_id, aeropuerto, prioridad):
        super().__init__()
        self.avion_id = avion_id
        self.aeropuerto = aeropuerto
        self.prioridad = prioridad
        self.pista_asignada = None
```

```
    def run(self):
        # Espera en la torre de control basado en la prioridad
        self.aeropuerto.torre_control.put((self.prioridad, self))
        self.aeropuerto.torre_control.get((self.prioridad, self))

        # Asignar pista
        self.aeropuerto.asignar_pista(self)

        # Simular despegue/aterrizaje
        time.sleep(random.uniform(1, 2))

        # Liberar pista
        self.aeropuerto.liberar_pista(self.avion_id, self.pista_asignada)

        # Notificar a la torre de control que ha terminado
        self.aeropuerto.torre_control.task_done()

# Simulación
aeropuerto = Aeropuerto()
aviones = [Avion(i, aeropuerto, random.randint(1, 3)) for i in range(10)]  # Diez aviones con prioridades aleatorias

for avion in aviones:
    avion.start()

for avion in aviones:
    avion.join()
```

Ejecución del Programa:

```
2023-11-07 04:45:29,418 - INFO - Avión 7 asignado a la pista 1.
2023-11-07 04:45:29,713 - INFO - Avión 6 liberó la pista 2.
2023-11-07 04:45:29,713 - INFO - Avión 8 asignado a la pista 2.
2023-11-07 04:45:29,788 - INFO - Avión 4 liberó la pista 0.
2023-11-07 04:45:29,788 - INFO - Avión 9 asignado a la pista 0.
2023-11-07 04:45:31,046 - INFO - Avión 8 liberó la pista 2.
2023-11-07 04:45:31,337 - INFO - Avión 7 liberó la pista 1.
2023-11-07 04:45:31,546 - INFO - Avión 9 liberó la pista 0.


...Program finished with exit code 0
Press ENTER to exit console.
```

Descripción del Entorno de Desarrollo:

Instrucciones para la ejecución del programa:

- Tener instalado Python (Versión: 3.x).
- Las bibliotecas utilizadas en el proyecto son parte de la librería estándar de Python. En caso de no contar con alguna de ellas, ejecuta el siguiente comando “pip install ‘nombre de la biblioteca’”.
- Copia el código en tu editor de texto, por ejemplo, Visual Studio Code.
- Una vez copiado y guardado el documento, abre tu terminal y dirígete al directorio donde guardaste el código, usando el comando “cd ”.
- Ya en el directorio donde tienes el archivo, ejecuta el siguiente comando “python MirandaVictorP_2.py” o si tienes varias versiones de python instaladas ejecuta “python3 MirandaVictorP_2.py”.
- En tu terminal verás una ejecución parecida a la mencionada anteriormente.

¿Qué lenguaje y qué versión se utilizaron?

R= Lenguaje: Python, Versión: 3 . 10 . 9

¿Bajo qué sistema operativo / distribución se desarrollo?

R = Sistema operativo: Linux, Distribución: Ubuntu, Versión: 22 . 04

Conclusión:

La simulación del manejo de tráfico en un aeropuerto demuestra de manera efectiva el uso de la programación concurrente para modelar y resolver problemas de la vida real que involucran recursos compartidos y actores múltiples. Utilizando semáforos, colas de prioridad y variables de condición, este programa asegura que los aviones (hilos) accedan a las pistas (recursos) de manera ordenada y segura, reflejando la necesidad de la exclusión mutua y la sincronización en entornos concurrentes.