Universidad Nacional Autónoma de México

Facultad de Ingeniería
Sistemas Operativos

Tarea 1: Los alumnos y el asesor

Grupo: 8

Tarea: 1

Integrante:

321101464 Michelle Ariana Castañeda González

1. Descripción general del problema

Durante el horario de atención del profesor, varios estudiantes pueden llegar a su cubículo para resolver dudas. El objetivo de esta simulación es modelar la interacción entre el profesor y los alumnos, de manera que la espera de todos sea lo más corta posible, respetando ciertas reglas de convivencia:

- El profesor cuenta con un número limitado de sillas en su cubículo. Cuando no hay alumnos que atender, las sillas funcionan como sofá y el profesor se acuesta a descansar.
- Los alumnos pueden tocar a la puerta en cualquier momento, pero no pueden entrar más alumnos de los que las sillas permiten.
- Para no confundir al profesor, solo un alumno puede presentar su duda y esperar la respuesta al mismo tiempo. Los demás alumnos deben esperar pacientemente su turno en las sillas disponibles.
- Cada alumno puede realizar entre 1 y varias preguntas, permitiendo que otros alumnos intercalen sus consultas.

2. Lenguaje y entorno de desarrollo

El programa fue desarrollado en Python 3.11 utilizando únicamente librerías estándar: `threading` para manejo de hilos y sincronización, `queue` para la cola FIFO de espera, `time` para controlar temporización y `random` para llegadas aleatorias de los alumnos.

El código es completamente portable y se ha probado en macOS. No requiere instalación de paquetes adicionales ni configuraciones externas.

3. Ejecución del programa

Para ejecutar el programa desde terminal:

```
python3 alumnos_asesor.py
```

El programa imprimirá en consola el estado de cada alumno y del profesor: llegada, espera, atención y finalización. La simulación termina cuando todos los alumnos han sido atendidos.

4. Estrategia de sincronización y mecanismos utilizados

Uso de semáforos

El semáforo se utiliza para coordinar el acceso entre el profesor y los alumnos, que son los hilos activos de la simulación. Se implementan dos semáforos principales:

- `profesor`: indica que hay al menos un alumno esperando. Cada vez que un alumno llega y hay sillas disponibles, ejecuta `profesor.release()` para despertar al profesor si está dormido (`profesor.acquire()` en el hilo del profesor). Esto asegura que el profesor solo intente atender alumnos cuando realmente hay alguien esperando.
- `alumno_listo`: permite que un alumno comience a ser atendido solo cuando el profesor está listo para recibirlo. Cuando el profesor toma al siguiente alumno de la cola, ejecuta `alumno_listo.release()` para que ese hilo de alumno pueda continuar. El alumno espera con `alumno_listo.acquire()`.

Uso de mutex (candado)

El mutex (`mutex = threading.Semaphore(1)`) protege el acceso a variables compartidas críticas:

- `alumnos_esperando`: número de alumnos actualmente en la sala de espera.
- `cola_espera`: estructura FIFO que mantiene el orden de llegada de los alumnos.

Uso de la cola FIFO

La cola FIFO (`queue.Queue`) garantiza que los alumnos sean atendidos en el orden de llegada, respetando la política "primero en llegar, primero en ser atendido". Cada vez que un alumno se sienta y entra a la cola, se mantiene su orden para el profesor, evitando que un alumno nuevo se "cuele" sobre otro que ya estaba esperando.

5. Refinamientos implementados

- Cola FIFO para respetar el orden de llegada.
- Atención del profesor más rápida para que la simulación no se demore.
- Se utiliza un contador de alumnos atendidos para finalizar correctamente la simulación y mostrar "Simulación finalizada".

6. Código fuente

```
import threading
import time
import random
from queue import Queue

SILLAS = 3
NUM_ALUMNOS = 10
cola_espera = Queue() # Cola FIFO para manejar el orden de llegada
   de los alumnos

profesor = threading.Semaphore(0)  # Sem foro que indica que hay un
    alumno esperando
alumno_listo = threading.Semaphore(0)  # Sem foro que indica que el
    alumno puede ser atendido
atendidos_lock = threading.Lock() #actualizar el contador de alumnos
    que entran
atendidos = 0  # Contador global de alumnos atendidos

# Funci n del profesor
def asesor():
    global atendidos
    while True:
        profesor.acquire()  # Espera a que llegue un alumno
        alumno_id = cola_espera.get()
        print(f"Profesor despierta para atender al alumno {alumno_id
           }.")
        alumno_listo.release()
        print(f"Profesor atendiendo al alumno {alumno_id}...")
```

```
        time.sleep(random.uniform(0.05, 0.2))
        print(f"Profesor termin  con el alumno {alumno_id}.")
        with atendidos_lock:
            atendidos += 1
            if atendidos == NUM_ALUMNOS_QUE_ENTRAN:  # Termina si todos los que entraron fueron atendidos
                break
    print("Todos los alumnos atendidos. Profesor se va a casa.")

# Funci n del alumno
NUM_ALUMNOS_QUE_ENTRAN = 0
NUM_ALUMNOS_LOCK = threading.Lock()

def alumno(id):
    global NUM_ALUMNOS_QUE_ENTRAN
    time.sleep(random.uniform(0.01, 0.1))
    if cola_espera.qsize() < SILLAS:
        cola_espera.put(id)
        with NUM_ALUMNOS_LOCK:
            NUM_ALUMNOS_QUE_ENTRAN += 1
        print(f"Alumno {id} lleg  y espera su turno. ({cola_espera.qsize()}/{SILLAS} sillas ocupadas)")
        profesor.release()
        alumno_listo.acquire()
        print(f"Alumno {id} est  siendo atendido.")
    else:
        print(f"Alumno {id} se fue, no hay sillas disponibles.")

# Programa principal
if __name__ == "__main__":
    print("Iniciando simulaci n...\n")

    hilo_profesor = threading.Thread(target=asesor)
    hilo_profesor.start()

    hilos_alumnos = []
    for i in range(NUM_ALUMNOS):
        t = threading.Thread(target=alumno, args=(i+1,))
        t.start()
        hilos_alumnos.append(t)

    for t in hilos_alumnos:
        t.join()

    hilo_profesor.join()
    print("\nSimulaci n finalizada.")
```

7. Conclusión

El programa utiliza mecanismos de sincronización adecuados (semáforos, mutex y cola FIFO) y respetando las reglas de interacción: un alumno por vez, límite de sillas, y atención en orden de llegada. La simulación es fácil de seguir, la salida es clara, y la implementación permite extenderse o refinarse según se requiera.