UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

Sistemas Operativos

Tarea 1

ALUMNOS:

Cuevas Quintana Amir

De la Rosa Flores Fernando

PROFESOR;

ING. GUNNAR EYAL WOLF ISZAEVICH

GRUPO 06

2025-1

- # Problema

Para esta tarea elegimos el problema de los alumnos y el asesor, el escenario consiste en modelar la interacción entre un profesor de facultad y varios estudiantes que buscan asesoría durante su horario de atención. El objetivo es que los alumnos dentro del cubículo del profesor puedan hacer preguntas de manera alternada, y este atienda a 1 a la vez, en caso de no haber alumnos esperando, las sillas del cubículo sirven como sofá y el profesor puede dormir en ellas, para este problema se tomó en cuenta que hay 3 sillas en el cubículo, cada uno tiene un límite de 3 preguntas y en caso de no haber sillas disponibles, los demás alumnos que lleguen no podrán hacer preguntas, se consideró un caso de llegada de 10 alumnos al cubículo.

- # ¿Cómo ejecutarlo y cosas que se requiere?

Lenguaje de programación: El proyecto está escrito en Java, seleccionamos este lenguaje debido a que en una materia previa en la facultad, se nos compartió que Java era excelente a la hora de trabajar problemas sobre concurrencia y quisimos explorar esta área, mencionado esto, necesitas tener instalado el Java Development Kit (JDK), versión 8 o superior, este puede ser descargado de Java Downloads | Oracle, para poder comprobar si está instalado se puede usar el comando '*java -version*' dando una salida similar a '*openjdk version "XX.XX.X" <other info>*'

Entorno de ejecución: Es importante contar con un sistema que soporte hilos, ya que este programa simula múltiples estudiantes y al profesor interactuando de forma simultánea mediante concurrencia.

Archivos necesarios:

- Asesor.java: Este archivo contiene la lógica de la interacción entre el profesor y los estudiantes.
- AsesorAlumnos.java: Este archivo ejecuta la simulación de varios estudiantes con diferentes tiempos de llegada y preguntas.

Sistema operativo: Cualquier sistema operativo que soporte Java (Windows, macOS, Linux).

Deberemos abrir la carpeta donde contengamos estos 2 archivos y abrir esta ubicación en nuestra terminal, así mismo deberemos revisar si ya esta compilado o tenemos que realizarlo, podemos hacerlo con la línea

-javac *.java Con el cual podremos compilar todos los archivos con esta extensión .java

```
PS C:\Users\fersp\Documents\codes\EJERCICIOS_JAVA\t1> javac *.java
```

Ahora si podremos ejecutar nuestro programa ejecutando nuestro archivo que contenga nuestro main, en este caso será el archivo "AsesorAlumnos.java" podremos ejecutarlo con la

siguiente linea, es importante considerar que esto está ejemplificado en el símbolo del sistema de Windows.

-java AsesorAlumnos

```
PS C:\Users\fersp\Documents\codes\EJERCICIOS_JAVA\t1> java AsesorAlumnos
```

Con esto sería suficiente para poder ejecutar nuestro código y ver el resultado

```
Asesor: Dormido, esperando estudiantes...
Alumno 1: *toc toc*, me voy a sentar, espero mi turno.
Alumno 1: Pregunta 1
Asesor: Despierto, listo para atender...
Alumno 2: *toc toc*, me voy a sentar, espero mi turno.
Asesor: Respuesta a alumno 1, pregunta 1
Alumno 2: Pregunta 1
Asesor: Respuesta a alumno 2, pregunta 1
Alumno 1: Pregunta 2
Alumno 3: *toc toc*, me voy a sentar, espero mi turno.
Asesor: Respuesta a alumno 1, pregunta 2
Alumno 3: Pregunta 1
Alumno 4: No hay sillas libres, me voy.
Alumno 5: No hay sillas libres, me voy.
Asesor: Respuesta a alumno 3, pregunta 1
Alumno 2: Pregunta 2
Alumno 6: No hay sillas libres, me voy.
Asesor: Respuesta a alumno 2, pregunta 2
Alumno 1: Pregunta 3
Asesor: Respuesta a alumno 1, pregunta 3
Alumno 3: Pregunta 2
Alumno 7: No hay sillas libres, me voy.
Asesor: Respuesta a alumno 3, pregunta 2
Alumno 2: Pregunta 3
Alumno 1: Terminó sus preguntas, deja la silla.
Alumno 8: *toc toc*, me voy a sentar, espero mi turno.
Alumno 9: No hay sillas libres, me voy.
Asesor: Respuesta a alumno 2, pregunta 3
Alumno 8: Pregunta 1
Asesor: Respuesta a alumno 8, pregunta 1
Alumno 3: Pregunta 3
Asesor: Respuesta a alumno 3, pregunta 3
Alumno 2: Terminó sus preguntas, deja la silla.
Alumno 10: *toc toc*, me voy a sentar, espero mi turno.
Alumno 10: Pregunta 1
Asesor: Respuesta a alumno 10, pregunta 1
Alumno 3: Terminó sus preguntas, deja la silla.
Alumno 10: Pregunta 2
Asesor: Respuesta a alumno 10, pregunta 2
Alumno 8: Pregunta 2
Asesor: Respuesta a alumno 8, pregunta 2
Alumno 10: Pregunta 3
Asesor: Respuesta a alumno 10, pregunta 3
Alumno 8: Pregunta 3
Asesor: Respuesta a alumno 8, pregunta 3
Alumno 10: Terminó sus preguntas, deja la silla.
Alumno 8: Terminó sus preguntas, deja la silla.
Asesor: Dormido, esperando estudiantes...
```

Debemos tener en cuenta que para esta solución requerimos lo siguiente

Simulación de hilos: El programa utiliza hilos para simular tanto al asesor como a los estudiantes. Cada estudiante es representado por un hilo que interactúa de manera concurrente con el asesor, siendo el profesor un recurso compartido.

Sincronización con semáforos: El programa utiliza semáforos para controlar el acceso a los recursos compartidos, como las sillas y el asesor. Estos semáforos garantizan que las interacciones sean ordenadas y que solo un estudiante a la vez pueda hacer preguntas.

Salida del programa: En la consola, se imprimirán mensajes que muestran el comportamiento del sistema en tiempo real: la llegada de estudiantes, si consiguen sentarse o deben irse, el proceso de preguntas, y las respuestas del asesor.

● Estrategia de sincronización

La sincronización del programa se realiza mediante el uso de semáforos, que permiten controlar el acceso a los recursos compartidos (como las sillas) y coordinar la interacción entre los estudiantes y el asesor. Mostraremos los elementos principales de la estrategia de sincronización:

1) **Semáforo para las sillas (semSillas):**

   Controla el acceso al número limitado de sillas disponibles en el cubículo del profesor. En el código, este semáforo se inicializa con un valor igual al número de sillas (numSillas = 3).

   Cuando un estudiante llega, intenta adquirir un permiso del semáforo para sentarse. Si hay una silla disponible, el semáforo decrece y el estudiante toma su lugar; si no hay sillas libres, el estudiante debe irse sin ser atendido. Una vez que un estudiante termina sus preguntas, libera el semáforo, lo que permite que otro estudiante ocupe la silla.

2) **Semáforo para el profesor (semProfesor):**

   Este semáforo se utiliza para controlar cuándo el profesor está dormido o despierto. Inicialmente, el profesor está dormido (es decir, el semáforo está en estado bloqueado). Cuando un estudiante llega y encuentra al profesor dormido, libera el semáforo para "despertar" al profesor.

   Si no hay estudiantes esperando (es dec**i**r, el semáforo no ha sido liberado), el profesor permanece dormido hasta que llegue alguien. Una vez despierto, el profesor atiende a los estudiantes uno por uno.

3) **Semáforo de turno (semTurno):**

   Este semáforo garantiza que solo un estudiante a la vez pueda hacer preguntas al profesor. Mientras un estudiante está formulando una pregunta, el semáforo bloquea el acceso para los demás estudiantes.

   Después de responder a una pregunta, el estudiante debe liberar el semáforo para permitir que otro estudiante formule su pregunta. Este mecanismo asegura que no haya confusión o superposición de preguntas y respuestas.

4) **Mutex (Exclusión mutua):**

   Se usa para controlar el acceso exclusivo al profesor y evitar que más de un estudiante interactúe con él a la vez. Al adquirir el mutex, un estudiante obtiene el derecho exclusivo de hacer preguntas al profesor. Luego, lo libera una vez que ha terminado su turno, permitiendo que otro estudiante pueda acceder al profesor.

Flujo de sincronización:

1) **Llegada de estudiantes:**

   Cuando un estudiante llega al cubículo, intenta sentarse en una de las sillas controladas por semSillas. Si hay espacio disponible, el estudiante se sienta; si no, se va sin ser atendido.

2) **Despertar al profesor:**

   Si un estudiante llega y encuentra al profesor dormido, libera el semáforo semProfesor, despertándolo para que pueda comenzar a atender. Solo el primer estudiante en la fila de espera puede despertar al profesor, gracias al uso del mutex.

3) **Realización de preguntas:**

   Cada estudiante puede hacer entre 1 y 3 preguntas. Después de cada pregunta, el semáforo semTurno se libera para permitir que otro estudiante haga su pregunta. Entre pregunta y pregunta, el estudiante debe ceder el turno, permitiendo que otros estudiantes también puedan ser atendidos.

4) **Terminar las preguntas:**

Cuando un estudiante ha terminado de hacer todas sus preguntas, libera su lugar en las sillas, lo que permite que otro estudiante ocupe su lugar. Esto se controla liberando el semáforo semSillas, aumentando la cantidad de permisos disponibles.

Esto nos permite que el sistema funcione de manera ordenada, justa y eficiente, permitiendo la atención secuencial de los estudiantes, mientras se evita que el profesor sea interrumpido o confunda las preguntas. La implementación de semáforos y exclusión mutua previene condiciones de carrera y garantiza que el comportamiento concurrente se mantenga controlado.

- # Refinamientos

- Se usan semáforos evitando así que haya inanición por diversas formas, ya que evitamos una espera indefinida, que 2 o más estudiantes pregunten al mismo tiempo y controlando quién puede sentarse y quien no.
- Con ayuda del semáforo y mutex nos aseguramos de que todos sean atendidos de una manera equitativa sin que ninguno acapare la atención del profesor.

- # Posibles mejoras

Más que algo que nos costara, creemos que quizás el problema podría serle agregado una cola de prioridad a los alumnos para simular por ejemplo cuando algunos tienen dudas un poco más urgentes que otros, simulando igualmente cuando un proceso tiene mayor prioridad que otro, igualmente creemos que quizás y podríamos encontrar alguna otra alternativa para evitar el uso de varios semáforos ya que coordinarlos en momentos puede llegar a ser algo complicado, escoger Java sin duda fue un reto ya que todo lo habíamos visto en Python, sin embargo es un lenguaje que se presta muy bien para resolver esta clase de problemas y un área de mejora sería también poder ver que ventajas nos ofrece resolver un problema sobre concurrencia ya sea en un lenguaje u otro.

- # Conclusión

El sistema de asesoría modelado en este proyecto demuestra la importancia de la sincronización en entornos concurrentes, como la interacción entre un profesor y varios estudiantes. Al usar hilos para simular el comportamiento concurrente de los estudiantes que llegan a hacer preguntas, se necesita un mecanismo eficiente para garantizar que las interacciones sean ordenadas y justas. Aquí es donde los semáforos y el mutex son muy utiles.

El mutex garantiza que solo un estudiante pueda interactuar con el profesor a la vez, evitando situaciones en las que varios estudiantes intenten hacer preguntas simultáneamente, lo que podría causar confusión. Por otro lado, los semáforos se utilizan para controlar los recursos

limitados, como las sillas disponibles en el cubículo, y para gestionar el estado del profesor (despierto o dormido), asegurando que solo sea despertado cuando hay estudiantes esperando.

Esta implementación proporciona una forma de manejar la concurrencia y coordinar los accesos a recursos compartidos en un sistema de múltiples hilos de manera coordinada, lo que es util para evitar conflictos y que todos los estudiantes en el cubículo puedan participar.