INTEGRANTES:
- García Peñafort Rodrigo.
- Santana Sánchez Gustavo.

Introducción

El problema planteado es una situación en la que somos roomies varios amigos (Paco, Yayo, Mau, Moi, Santana y Roy) y llega la hora de cocinar. Básicamente tenemos que compartir la estufa, sartenes, ollas, la licuadora, el microondas y un tostador. Nos tenemos que poner de acuerdo, por ejemplo, para usar de forma adecuada la estufa, ya que las ollas pueden abarcar el espacio de otro quemador y entonces alguien más tendría que esperar hasta que se libere ese espacio. También podría pasar que dos personas están intentando usar la licuadora al mismo tiempo, por lo que requerimos de coordinación.

Sin coordinación la concurrencia puede resultar incluso peor que la forma secuencial de hacer las cosas, es decir, podríamos tardar más en cocinar que si sumaramos el tiempo que cada uno tardaría si cocina en un momento distinto. En este sentido, si hay dos o más roomies que quieren cocinar al mismo tiempo, entonces los eventos que se pueden presentar son las condiciones de carrera (no se ponen de acuerdo para acceder a determinado recurso); un uso inadecuado del recurso (alguien está usando el microondas y llega alguien más y lo abre); alguien obtiene un sartén u olla e intenta acceder a la estufa, pero no hay suficiente espacio.

Por otra parte, existen eventos en los que el orden en que ocurren no afectan el resultado: cuando alguien está ocupando la licuadora, pero alguien más se encuentra usando el tostador; alguien está usando la estufa y alguien más el microondas. Básicamente cuando se encuentran haciendo diferentes acciones jamás va a afectar el resultado final porque no intervienen directamente con las acciones de alguien más.

Mecanismos de sincronización empleados

Se emplearon dos mecanismos de sincronización del paquete *java.util.concurrent*:

- ***Semaphore***: controla el acceso a recursos compartidos con un cupo limitado. Se utilizó para la estufa, ollas y sartenes.
- ***Reentrantlock***: controla el acceso exclusivo a ciertos recursos. Se utilizó para el tostador, la licuadora, el microondas y el lavaplatos.

De esta forma se garantiza que no haya más roomies (hilos) cocinando de lo que la estufa permite y solo una persona puede usar un electrodoméstico a la vez.

Lógica de operación

Cada hilo representa a un roomie que quiere preparar su comida. Estos roomies pueden acceder a los siguientes recursos de manera compartida:

- private static final Semaphore estufa;
- private static final Semaphore sarten;
- private static final Semaphore olla;
- private static final Lock tostador;
- private static final Lock microondas;
- private static final Lock licuadora;
- private static final Lock lavaplatos;

El roomie genera su rutina de cocina al azar, que siempre empezará por *preparar* y *cocinar,* después, puede hacer de una a tres actividades adicionales (usar el microondas, la licuadora o el tostador) y por último debe lavar sus utensilios. Cuando una acción requiere sincronización se realiza lo siguiente:

- Se obtiene el recurso usando *acquire()*, para semáforos, o *lock()*, para recursos de accesos exclusivo.
- Para simular que se está usando el recurso se emplea *Thread.sleep()* y dependiendo de la acción puede tardar más o menos tiempo.
- Por último se libera el recurso mediante *release(),* para semáforos, o *unlock(),* para los recursos de accesos exclusivo.

Además, se muestran los pasos con una función llamada *logEvento,* la cual se comunica con la interfaz gráfica. La información que ésta recibe es el estado, el nombre y la acción del hilo (roomie) actual. Por otra parte, la interacción entre hilos se da únicamente a través del acceso a recursos compartidos. No se comunican directamente entre sí.

Descripción del entorno de desarrollo, suficiente para reproducir una ejecución exitosa

Esta simulación fue desarrollada y probada en el siguiente entorno:

- Lenguajes: Java (JDK 21.0.6), Python (3.13.2).
- Bibliotecas en Java:
  - java.util.concurrent.ExecutorService
  - java.util.concurrent.Executors
  - java.util.concurrent.Semaphore
  - java.util.concurrent.locks.Lock
  - java.util.concurrent.locks.ReentrantLock
  - java.util.Random
  - java.util.List
  - java.util.Collections
  - java.util.ArrayList
  - java.util.Arrays
  - java.util.concurrent.TimeUnit

- Bibliotecas en Python:
    - tkinter (versión 8.6)
    - tkinter.ttk
    - subprocess
    - threading
    - collections.defaultdict

Para instalar la biblioteca tkinter (en Fedora) se puede emplear el siguiente comando:

sudo dnf install python-tkinter

- Sistema operativo:
    - Fedora Linux 41 (Workstation Edition)

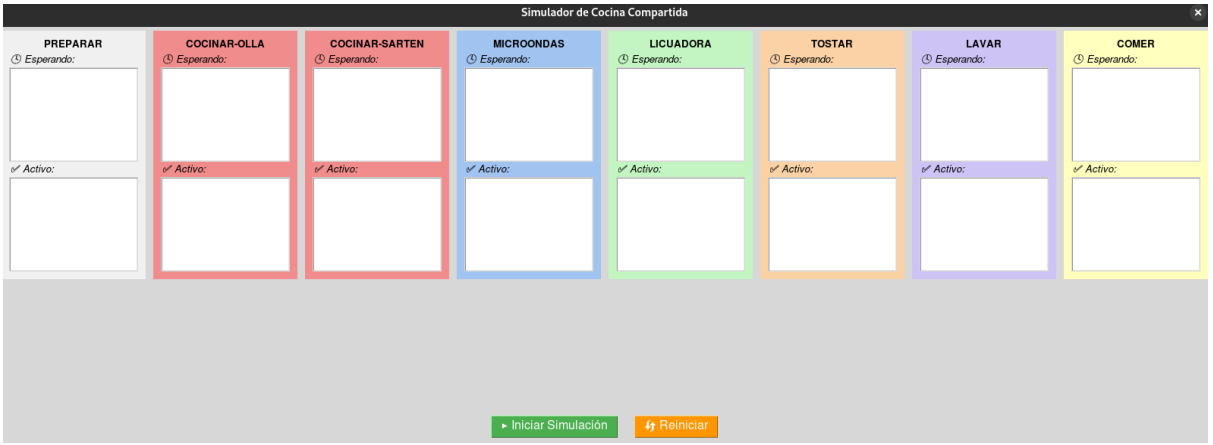
P*antallazos* de una ejecución exitosa.

- Invocación:

Es necesario compilar el archivo con extensión java. Para correr el programa se ejecuta la segunda instrucción que se observa en la siguiente imagen:

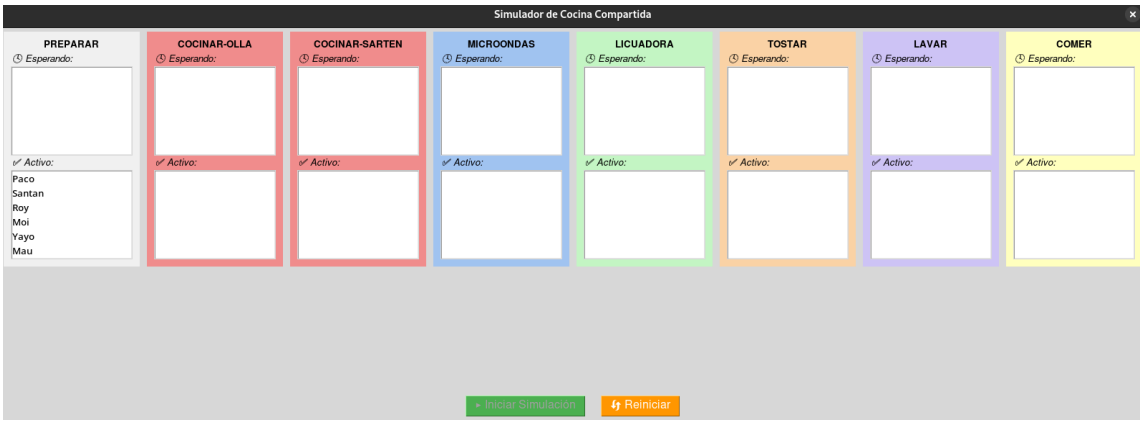
```
javac CocinaCompartida.java
python interfazCocinaCompartida.py
```

- Interfaz sin iniciar el programa aún:

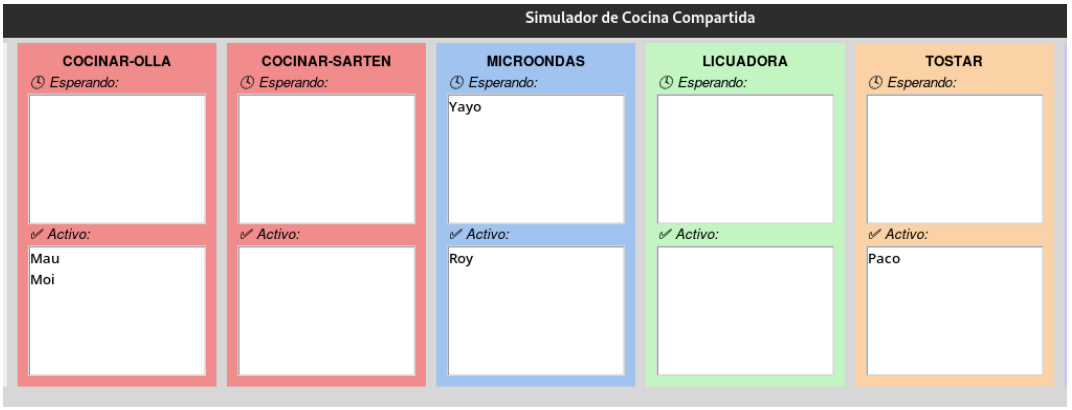


En este momento es necesario dar click sobre el botón que dice “iniciar simulación”.

- Inicia el programa y todos empiezan a preparar sus ingredientes:



- Transcurso de la ejecución



- La mayoría se encuentran en el área de lavar, por lo que es el último paso antes de terminar

- Final de la ejecución

- Más ejemplos de cómo finaliza la ejecución

Una vez alcanzado este momento, se desbloqueará el botón que dice ‘iniciar simulación’. Si se desea ejecutar de nuevo el programa, entonces es necesario presionar el botón ‘reiniciar’ y, luego, el botón ‘iniciar’.