

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

			PRÁCTICA DE LABORATORIO		
CARRERA: COMPUTACION			ASIGNATURA: COMPUTACION PARALELA		
NRO. PRÁCTICA:	4	TÍTULO PRÁCTICA: Desarrollo e implementación de aplicaciones de cómputo paralelo basado procesos			
OBJETIVOS					
<ul style="list-style-type: none">Diseñar e implementar aplicaciones en paralelo usando las principales tecnologías					
INSTRUCCIONES		Enunciado			
		En esta práctica se explorará el uso de la librería multiprocessing en Python para resolver un problema de procesamiento concurrente utilizando POO (clases). El caso de estudio se basa en la simulación de un sistema de <i>gestión de pedidos en un restaurante</i> , donde distintos procesos deben gestionar tareas como la recepción de pedidos, su preparación, y la actualización del inventario . Se debe implementar una solución que utilice clases para <i>organizar la lógica, sincronizar datos entre procesos y permitir el intercambio de información</i> .			
		Contexto			
		El restaurante tiene los siguientes componentes principales:			
		<ol style="list-style-type: none">Recepción de pedidos: Los clientes realizan pedidos que se registran en un sistema.Preparación de pedidos: Los pedidos se procesan en orden, y se registra el estado del inventario.Inventario: El inventario se actualiza en tiempo real con cada pedido preparado.			
		Tu tarea es crear una aplicación que simule este sistema utilizando la librería multiprocessing .			
		Requerimientos funcionales			
		General			
		<ol style="list-style-type: none">Estructura basada en clases: Implementar la lógica utilizando clases que representen las distintas funcionalidades (Recepción, Preparación, Inventario).Sincronización de datos: Garantizar que los datos sean consistentes utilizando mecanismos de sincronización como Barrier con Lock o Semaphore.Intercambio de datos: Utilizar colas (Queue) para la comunicación entre procesos.			
		Detalles específicos			
		<ol style="list-style-type: none">Crear una clase Pedido para representar un pedido, que contenga atributos como:<ol style="list-style-type: none">ID del pedido.Lista de ítems solicitados.			

- c. Estado del pedido (pendiente, en preparación, completado).
2. Crear una clase **GestorPedidos** con los siguientes métodos:
 - a. **recibir_pedido(self, pedido: Pedido)**: Agrega un nuevo pedido a la cola.
 - b. **procesar_pedido(self)**: Extrae un pedido de la cola y lo marca como "en preparación".
3. Crear una clase **Inventario** con funcionalidades como:
 - a. **actualizar_inventario(self, item: str, cantidad: int)**: Actualiza el inventario al preparar un pedido.
 - b. **consultar_inventario(self)**: Devuelve el estado actual del inventario.
4. Implementar el flujo principal:
 - a. Crear procesos para la recepción y preparación de pedidos.
 - b. Utilizar colas para pasar los pedidos entre los procesos.
 - c. Usar bloqueos para evitar inconsistencias en el inventario.
 - d. Simulación del flujo:
 - a. Crear un conjunto inicial de pedidos simulados que se agregan a la cola.
 - b. Procesar los pedidos en orden utilizando múltiples procesos de preparación.
 - c. Actualizar el inventario tras cada pedido preparado.
 - d. En caso de que el inventario no tenga suficiente stock para un pedido, registrar el pedido como "no procesado" y continuar con el siguiente.
 - e. Generar un reporte final (en la consola) que incluya:
 - a. Pedidos completados exitosamente.
 - b. Pedidos no procesados por falta de inventario.
 - c. Estado final del inventario.

Guía de evaluación (5 puntos totales)

1. Uso adecuado de clases y métodos (1.5 pts):
 - La solución organiza correctamente la lógica en clases.
 - Los métodos implementan la funcionalidad requerida de manera modular.
2. Sincronización de procesos (1.5 pts):
 - Se utilizan correctamente mecanismos como **Lock** o **Semaphore** para evitar conflictos de acceso a los datos.
 - El inventario refleja los cambios de manera consistente.
3. Intercambio de datos entre procesos (1 pt):
 - Los procesos se comunican correctamente utilizando colas (**Queue**).
 - Los pedidos se transfieren entre procesos sin errores.
5. Resultados y ejecución (1 pt):
 - La aplicación cumple con el flujo especificado.
 - La simulación se ejecuta sin errores y genera resultados lógicos.

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

ACTIVIDADES POR DESARROLLAR
1. Diseñar e implementar las clases
2. Implementación de procesos y sincronización
3. Simulación del flujo principal
4. Generar el informe de la práctica con el desarrollo de cada uno de los puntos descritos anteriormente
5. Subir al AVAC el informe del proyecto en formato *.pdf. El informe debe contar con conclusiones apropiadas y la firma de cada estudiante
RESULTADO(S) OBTENIDO(S): <ul style="list-style-type: none"> Diseña e implementa aplicaciones en paralelo.
CONCLUSIONES: <ul style="list-style-type: none"> Los estudiantes podrán interactuar con el análisis y diseño de algoritmos paralelos
RECOMENDACIONES:

Docente: Ing. Gabriel León Paredes, PhD.



Firma: