LICENCIATURA EN CIENCIA DE DATOS
Computación Concurrente 2022-I
Segundo Examen Parcial

Fecha: 19 de noviembre 2022

Profesor: Oscar A. Esquivel Flores
Cubículo: IIMAS-325
e-mail: oscar.esquivel@iimas.unam.mx

Objetivo:

El propósito del examen consiste en poner en práctica los conceptos de cómputo concurrente y los mecanismos de sincronización vistos en clase.

Generalidades:

1. La implementación de la concurrencia será por medio de hilos utilizando `threading` del lenguaje de programación Python.
2. Utiliza el material visto en clase y algunas fuentes de información adicionales que consideres importantes, **agrega la referencia correspondiente**.
3. Incluye comentarios al código a manera de documentación.

Descripción:

Lee con detenimiento y precaución los siguientes actividades, discute con tu equipo su debida comprehensión y propuesta de solución:

1. Implementa un programa secuencial que calcule el producto de dos matrices (considerablemente grandes, sugerencia $10,000 \times 10,000$). Posteriormente refactoriza el código de tal forma que utilices concurrencia, por medio de hilos, para que realicen esta tarea de manera conjunta, cada hilo calculará un subconjunto del producto. Modifica la versión concurrente para hacer un **análisis de rendimiento: speed-up y eficiencia**, incluye el procedimiento que elabores para el cómputo de los tiempos y métricas, así como las gráficas correspondientes; no olvides variar el tamaño de las matrices y el número de hilos. El programa lo nombrarás **Programa1**.
2. Implementa una **simulación de la fábula** que cuenta la carrera entre la liebre y la tortuga. Para hacerlo más interesante la carrera será cuesta arriba por una pista resbaladiza, de modo que a veces podrán resbalar y retroceder algunas posiciones. Habrá un *thread* (instanciado con clases) que implementará la tortuga y otro la liebre. Cada uno se suspenderá durante un segundo y luego evaluará lo que ha pasado según unas probabilidades:

animal	suceso	probabilidad	movimiento
Tortuga	Avance rápido	50 %	3 casillas a la arriba
	Resbaló	20 %	6 casillas a la abajo
	Avance lento	30 %	1 casilla a la arriba
Liebre	Duerme	20 %	
	Gran salto	20 %	9 casillas a la arriba
	Resbalón grande	10 %	12 casillas a la abajo
	Pequeño salto	30 %	1 casilla a la arriba
	Resbalón pequeño	20 %	2 casillas a la abajo

Calcula la probabilidad con random, de 1 a 100 y determina con dicho número qué ha hecho cada animal. Considera que hay 70 casillas, de la 1 a la 70, la 1 de salida y la 70 de llegada. Si resbala al principio

vuelve a la 1, nunca por debajo. Tras cada segundo y después de calcular su nueva posición escribe una línea por cada animal, con espacios en blanco de 1 la *posición-1* y luego una letra T para la tortuga y una L para la liebre. Imprima al comienzo de la carrera un mensaje. Después de imprimir las líneas determine si alguno ha llegado a la meta y ha ganado, escribiendo un mensaje. Si ambos llegan a la vez declare un empate. El programa lo nombrarás **Programa2**.

3. ¿Recuerdas el problema del **productor-consumidor**? Implementa la solución a este problema con un productor-un consumidor, mediante semáforos.

   a) La primera cuestión a considerar es que se ha de controlar el acceso exclusivo a la sección crítica, es decir, al propio buffer. La segunda cuestión importante reside en controlar dos situaciones: i) no se puede insertar un elemento cuando el buffer está lleno y ii) no se puede extraer un elemento cuando el buffer está vacío. En estos dos casos, será necesario establecer un mecanismo para bloquear a los procesos correspondientes.

   b) Un acercamiento a la solución se basa en los siguientes elementos:

   - **buffer**: Semáforo binario para proporcionar exclusión mutua para el acceso al buffer de productos. El semáforo se inicializa en 1.
   - **empty**: Semáforo contador que se utiliza para controlar el número de posiciones vacías del buffer. Este semáforo se inicializa a n, siendo n el tamaño del buffer.
   - **full**: Semáforo contador que se utiliza para controlar el número de posiciones llenas del buffer. Este semáforo se inicializa a 0.

   c) En esencia, los semáforos **empty** y **full** garantizan que no se puedan insertar más elementos cuando el buffer esté lleno o extraer más elementos cuando esté vacío, respectivamente. Por ejemplo, si el valor interno de **empty** es 0, entonces un proceso que ejecute `wait/acquire` sobre este semáforo se quedará bloqueado hasta que otro proceso ejecute `signal/release`, es decir, hasta que otro proceso produzca un nuevo elemento en el buffer.

   Al programa lo nombrarás **Programa3**

Entrega:

- Los tres programas deberán estar editados en un notebook de jupyter, ejecutándose de manera adecuada y sin mensajes de error.
- Nombra el notebook **ex2_CC-2022_I_#equipo** y agrega en una celda markdown el nombre completo de los integrantes.
- **Un solo** integrante del equipo colocará el examen en el canal #examen del servidor de nuestro curso en Discord.
- El notebook con las implementaciones correspondientes deberá estar publicado a más tardar el **martes 23 de noviembre a las 23:59 hrs**.

Comentarios:

- No utilices conceptos que no se hayan visto en clase.
- Puedes consultar todo el material que consideres necesario, por supuesto código, agrega la debida referencia y/o bibliografía.
- Agrega comentarios generales en celdas markdown que expliquen o documenten el funcionamiento de tus implementaciones y también incluye comentarios al código para aclarar variables, procedimientos, acciones etc. No olvides seguir las mejores prácticas de programación.
- Esfuérzate en desarrollar tu propio código y no lo copies totalmente.
- Se recomienda organización en el equipo, trabajo equitativo y diálogo cordial.
- Abrimos el canal #**noentiendo** en el servidor de Discord para externar dudas, preguntas, ansiedades, congojas y/o tristezas que está disponible desde el inicio del examen hasta la fecha y hora de la entrega.