Marco Ferraro Rodriguez | B82957 | CI0177
Programación Paralela y Concurrente
Examen 1

---

1. Respuesta Breve : [dna sequences] (20pts)

1.1. Describa brevemente cuál fue la sección (o secciones) del programa que resolvió utilizando programación paralela, indicando el o los métodos de sincronización utilizados y la zona crítica.

1.1.1. Método Paralelo

Para resolver el problema en de [DNA sequences] se utiliza una sección paralela donde el método paralelo para uso de cada thread se llama **process_dna(void*args)**. Lo importante de destacar de este método son 2 secciones importantes.

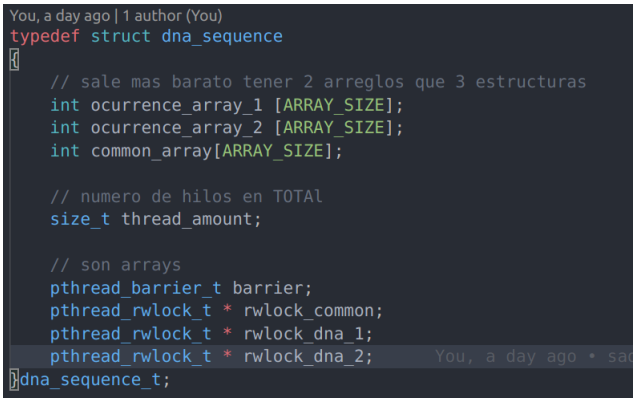
```
if(my_dna == 1){
    for(size_t i = my_thread_id; i < sizeof(DNA_SEQUENCE_1) - 1; i = i + jump){
        pthread_rwlock_wrlock(&shared_data->rwlock_dna_1[0]);
            shared_data->ocurrence_array_1[DNA_SEQUENCE_1[i] - 'a']++;
        pthread_rwlock_unlock(&shared_data->rwlock_dna_1[0]);
    }
}else{
    for(size_t i = my_array; i < sizeof(DNA_SEQUENCE_2) - 1; i = i + jump){
        pthread_rwlock_wrlock(&shared_data->rwlock_dna_2[DNA_SEQUENCE_2[i] - 'a']);
            shared_data->ocurrence_array_2[DNA_SEQUENCE_2[i] - 'a']++;
        pthread_rwlock_unlock(&shared_data->rwlock_dna_2[DNA_SEQUENCE_2[i] - 'a']);
    }
}
```

Para calcular las ocurrencias dentro de el vector de ADN, cada hilo empieza en una posición diferente (si es que hay mas de 1 hilo). Dependiendo de que cadena de ADN (1 o 2) le corresponda al hilo, va a analizar un carácter especifico del **su vector de ADN**. El vector de ocurrencias es un vector de enteros, entonces, por ejemplo si un hilo encuentra la letra 'a' en la posición correspondiente de contador para la letra 'a' se sumara 1 valor. En este for se dan saltos para que cada hilo no analice un carácter que otro hilo pueda analizar.

```
// Inicio de la interseccion
pthread_barrier_wait(&shared_data->barrier);
for(size_t i = my_thread_id; i < ARRAY_SIZE; i = i + big_jump){
    pthread_rwlock_rdlock(&shared_data->rwlock_dna_1[i]);
    pthread_rwlock_rdlock(&shared_data->rwlock_dna_2[i]);
    if(shared_data->ocurrence_array_1[i] != 0 && shared_data->ocurrence_array_2[i] != 0){
        pthread_rwlock_wrlock(&shared_data->rwlock_common[i]);
            shared_data->common_array[i]++;
        pthread_rwlock_unlock(&shared_data->rwlock_common[i]);
    }
    pthread_rwlock_unlock(&shared_data->rwlock_dna_1[i]);
    pthread_rwlock_unlock(&shared_data->rwlock_dna_2[i]);
}
```

La otra sección del código paralelo que cabe destacar es el calculo de la intersección o letras comunes. Lo que ocurre acá es que los hilos se acomodan en posiciones diferentes dependiendo de su id y comparan que en la casilla correspondiente de carácter, sea 'a', exista un valor diferente de 0 en los 2 vectores de ocurrencias. Se suma 1 al valor del carácter correspondiente en el vector **common_array**.

1.1.2. Seccion Critica



Esta es la sección critica, que se compone de 3 arreglos de enteros, la variable que identifica la cantidad de hilos y los mecanismos de sincronización.

1.1.3. Métodos de Sincronización

Para los métodos de sincronización se usan 3 **rwlock_dna**, uno para cada vector de enteros. El punto de tener 3 **rwlock_dna** y no 1 es para reducir la serialización del proceso paralelo, además de tenerlo como un mecanismo de seguridad contra una posible condición de carrera o corrupción de datos.

1.2. Con respecto a los enfoques vistos en clase: optimización y separación de asuntos, ¿Cuál es el enfoque en el que se centra este problema? Explique.

El enfoque de este problema es separación de asuntos. Ya que se quiere resolver este problema con n cantidad de hilos, y dependiendo de ese n los hilos deben procesar diferentes entradas de las cadenas o arreglos de adn.

1.3. ¿Considera que la solución paralela es siempre la mejor para resolver este tipo de problemas? Explique.

En este caso depende. En lo personal paralelizar este procesamiento de datos tiene mas sentido ya que se están distribuyendo procesos o asuntos, sin embargo la sintaxis si se hizo mas complicada y en la depuración aparecieron errores de usos de memoria que no hubieran salido si se hubiera implementado de forma serial. Por lo tanto, depende del problema. Si se quiere optimizar el rendimiento y hacer procesos mas rápidos hay que tener en cuenta el overhead que genera los métodos de sincronización. Pero si se ve que el problema se entiende mas de forma paralela por separación de asuntos también hay que considerarlo. Además, en el caso de este problema, si se piden muchísimos hilos la computadora no es posible de asignar todos los recursos para el proceso.

2. Respuesta Breve : [sushiBarProblem] (10pts)

2.1. Describa cuál fue el cambio realizado. Explique brevemente si considera que el cambio realizado es mejor o peor en algún aspecto. Si considera que no hay diferencia, explique por qué

Con respecto a la pregunta único que mejoro fue en la comprensión de la sintaxis y lógica del código. Por la naturaleza del problema. Sin embargo, esto también puede ocasionar un problema.

En mi caso me costo transformar varios mecanismos de sincronización a variables de condición.

Sin embargo, también considero que, a como se resolvió este problema, se pierde tiempo de ejecución, ya que para usar una variable de condición se tuvo que remplazar el uso de un semáforo, y el overhead de usar una variable de condición es mayor ya que se usa la variable de condición y un mutex.