Proyecto 2

Profesora
Cecilia Hernández

Ayudante
Jesús Gomez

Integrantes
Jaime Ansorena Carrasco - 2020401497
Diego Oporto Valenzuela - 2020430403
Dayan Sáez Cubillos - 2020444854

Descripción del código

Se implementó un código que permite la lectura de archivos contenidos en un directorio especifico. Dichos archivos corresponden a genomas, donde estos son leídos linea a linea para extraer el porcentaje de contenido genético de CG en un genoma (un archivo), esto es $\frac{C+G}{A+C+T+G}$. A partir de ese porcentaje se seleccionan los genomas que poseen una proporción de CG mayor a un *umbral* dado por parámetro, dicho *umbral* está restringido a: $0 \leq umbral \leq 1, umbral \in \mathbb{R}$. Cada archivo es procesado concurrentemente usando hebras, en donde cada hebra procesa un archivo distinto. Aquellos archivos que cumplen con la condición son identificados y se almacena el nombre del genoma en una cola compartida (FIFO). Para evitar condiciones de carreras, las secciones críticas son tratadas con exclusión mutua, por medio de variables de condición o semáforos, el usuario, al momento de iterar elije que mecanismo de sincronización usar.
A continuación, se describen las funciones implementadas:

- `float calcularCG(string fileName)`

  Calcula el contenido de CG en un genoma. Se utiliza la función `ifstream()` para acceder al archivo del cual se recibe su nombre por parámetro. Luego el archivo es recorrido por linea, ignorando las lineas que comienzan con el carácter '>'. Se cuenta la cantidad de C y G presentes en el genoma y la cantidad de bases nitrogenadas totales. Al iterar todo el archivo se retorna el porcentaje de CG en el genoma.

- `void leerDirectorio(string directorio, vector<pair<string,string>> &genomas)`

  Lee el contenido de un directorio específico utilizando la biblioteca `dirent.h`. Primero abre el directorio con `opendir` e itera sobre cada entrada utilizando `readdir`. Para cada archivo, agrega un par al vector genomas que contiene la ruta del directorio y el nombre del archivo. Luego, cierra el directorio con `closedir`. En caso de error al abrir el directorio, o si se proporciona un directorio que no existe, se imprime un mensaje de error y se termina el programa.

- `int main()`

  Recibe como argumentos un directorio y un umbral, luego, se lee el contenido del directorio con la función `leerDirectorio()`, obteniendo un vector de pares que representan la ruta y nombre de los archivos genómicos. Posteriormente, se le solicita al usuario que elija entre usar una cola compartida protegida por mutex y variable de condición o una protegida por semáforos. Dependiendo de la elección, se instancia la clase correspondiente (`ColaCompartidaMutex` o `ColaCompartidaSemaforo`) y se inician las hebras para procesar los genomas e imprimir los resultados.

Cada hebra creada ejecuta la función `pushGenoma()` de la instancia de la cola compartida correspondiente. Una vez que todas las hebras han terminado su ejecución, se activa la señal de terminación con la función `terminar()` en la cola compartida y se espera a que la hebra de impresión finalice antes de que el programa termine.

Por otro lado, el código cuenta con dos clases, `ColaCompartidaMutex` y `ColaCompartidaSemaforo`, que representan las colas compartidas implementadas con mutex y con semáforos.

En ambas clases se definieron los mismos métodos, pero se diferencian en que una utiliza mutex y variable de condición, y la otra utiliza un semáforo:

- `void pushGenoma(string directorio, string archivo, float umbral)`

  Calcula la proporción de bases nitrogenadas C y G en un archivo genómico llamando a `calcularCG()`, si esta proporción supera el umbral especificado, agrega el nombre del archivo y la proporción calculada a la cola compartida. En `ColaCompartidaMutex`, se utiliza un mutex y una variable de condición para garantizar la exclusión mutua y notificar a las hebras que están esperando. Por otro lado, en `ColaCompartidaSemaforo`, se utiliza un semáforo para controlar el acceso concurrente a la cola.

- `void printGenoma()`

  Imprime los genomas almacenados en la cola compartida. En la implementación con `ColaCompartidaMutex`, se utiliza un mutex y una variable de condición para garantizar la exclusión mutua y esperar a que haya elementos en la cola antes de imprimir. La condición de espera en `ColaCompartidaMutex` verifica que la cola no esté vacía o que la señal de terminación no haya sido activada. En la implementación con `ColaCompartidaSemaforo`, el método verifica si la cola no está vacía antes de adquirir y liberar el semáforo.

- `void terminar()`

  Señaliza la finalización del procesamiento de genomas, permitiendo una terminación ordenada de las hebras. En la implementación con `ColaCompartidaMutex`, se utiliza un mutex para asegurar la operación de actualización de la variable booleana `terminado` y se notifica a través de una variable de condición a las hebras que están esperando en la función `printGenoma`. Por otro lado, en la implementación con `ColaCompartidaSemaforo`, el método simplemente actualiza la variable `terminado` sin necesidad de un mutex, ya que la operación de escritura es atómica con respecto a los semáforos.

Evaluación Experimental

Se realizaron las pruebas utilizando los directorios proporcionados. Por cada directorio se ejecutaron iteraciones con un umbral fijo, en donde se comparan los tiempos y espacio mediante el Maximum Resident Set entre usar variables de condición o semáforos.

Directorio	genoma	bacteria	invertebrate	sample
Variable de condición (s)	8.47	13.66	194.73	3.71
Semáforo (s)	8.69	14.43	275.65	4.58

Cuadro 1: Tiempo tomado en analizar los archivos para un directorio con un umbral de 0.5

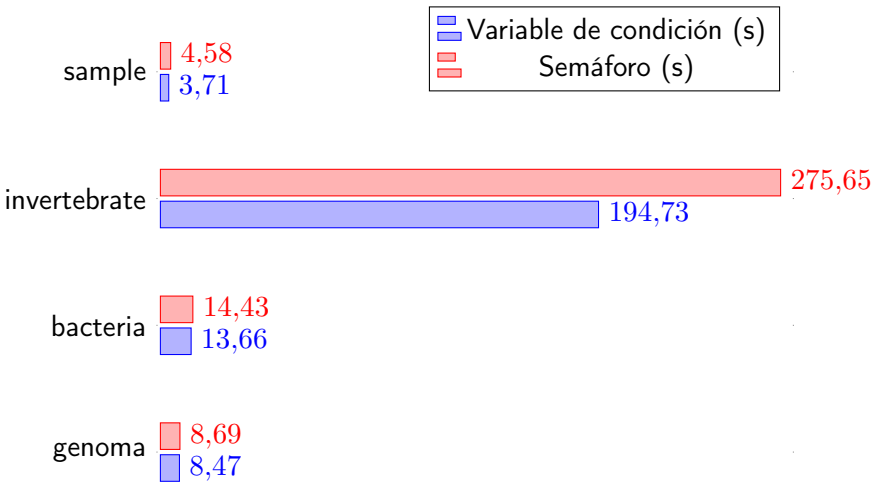

Figura 1: Análisis Temporal

Directorio	genoma	bacteria	invertebrate	sample
Variable de condición (kbytes)	9952	10296	16308	4244
Semáforo (kbytes)	10008	14156	14208	6240

Cuadro 2: Maximum Resident Set por mecanismo de sincronización para un un directorio con umbral=0.5

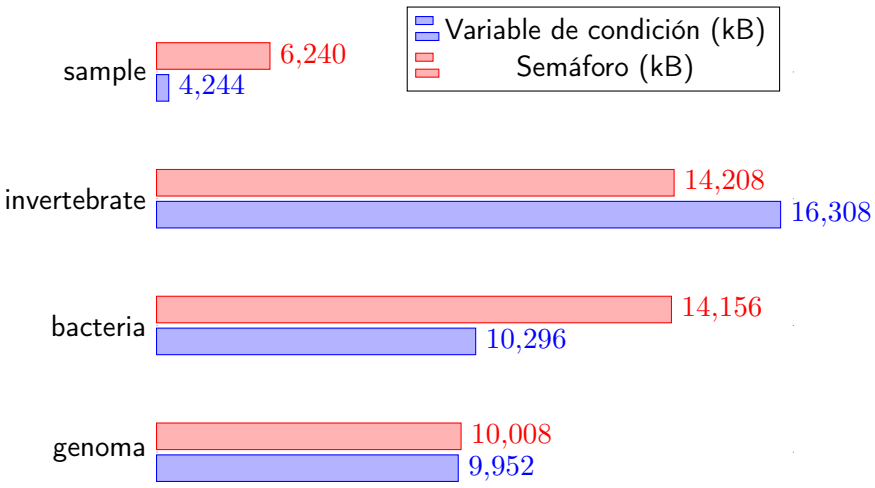

Figura 2: Análisis Espacial

Conclusiones

Se aprecia que al utilizar variables de condición los tiempos son menores, esto puede ser debido a la naturaleza del problema, el cual para evitar condiciones de carrera se basta con usar un mutex, por lo que el contador que posee el semáforo no es imprescindible en este caso.

Al comparar el espacio utilizado también se ve que usar variables de condición es más conveniente. La diferencia de espacio se puede deber a la estructura utilizada para implementar los semáforos, la cual posee un contador y una cola de punteros a las hebras. Sin embargo, para el directorio invertebrates el resultado difiere, a pesar de iterar varias veces, por lo que para directorios con muchos archivos puede ser más conveniente utilizar semáforos si lo que se quiere es priorizar el uso de espacio.

Los resultados indican que utilizar variables de condición es lo más eficiente, sobre todo en términos de tiempo, pero, si se tienen archivos muy grandes y es necesario priorizar el espacio, es mejor opción usar semáforos.