Ejercicio 2

Enunciado

Implementar una función ConcurrentHashMap count words(string arch) que tome un archivo de texto y devuelva el ConcurrentHashMap cargado con las palabras del archivo. Tomamos como palabras las separadas por espacio. Implementarlo de manera no concurrente.

Implementación

Algoritmo ConcurrentHashMap countwords(string arch)

1: ConcurrentHashMap map

2: Mientras queden palabras en arch

3: map.addAndInc(palabra)

4: return map

No se debía utilizar procesamiento en paralelo, por lo que no se incluyeron sincronizaciones.

Ejercicio 3

Enunciado

Implementar una función ConcurrentHashMap count words(list<string>archs), que tome como parámetro una lista de archivos de texto y devuelva el ConcurrentHashMap cargado con esas palabras. Deberá utilizar un thread por archivo.

Implementación

Para resolver este ejercicio, utilizamos

Estructura datos\_thread: Almacena los datos del thread

Estructura datos\_thread {

unsigned int thread\_id;

ConcurrentHashMap\* map;

string archivo;

};

Función ArmoHashMap

Cada thread deberá ejecutar una función llamada ArmoHashMap, que armará el hashmap.

Primero creamos un ConcurrentHashMap y luego iteramos por cada elemento de la lista.

Por cada uno de estos ítems, creamos un thread que es el que se encargará de recorrer todo el archivo de texto.

Su deber será agregar las palabras al ConcurrentHashMap.

ArmoHashMap

1: struct datos\_thread\* datos;

2: string arch

3: open arch

4: mientras queden palabras en arch

5: (datos->map)->addAndInc(palabra)

6: close arch

7: pthread exit(NULL);

count words(list<string> archs)

1: número threads archs.size()

2: pthread t threads[número threads]

3: datos thread args[número threads]

4: ConcurrentHashMap map

5: para cada elemento en archs

Setting del próximo thread. Se almacena el archivo a recorrer en args[i] donde i es el índice del elemento actual.

creación de thread (seteado en punto6). Llama a la función ArmoHashMap

6: n para

7: Espera a que todos threads completen la ejecución

8: Return map

Ejercicio 4

Enunciado

Implementar una función ConcurrentHashMap count words(unsigned int n,

list<string>archs) igual a la del punto anterior, pero utilizando n threads, pudiendo ser

n menor que la cantidad de archivos.

No puede haber threads sin trabajo si aún hay archivos por procesar.

Implementación

Para la implementación de este ejercicio, utilizamos

struct datos\_tread\_ejercicio4

int thread\_id;

ConcurrentHashMap\* map;

iterator\* itera\_ini;

iterator itera\_fin

pthread\_mutex\_t\* mutex\_itera;

Función ArmoHashMapEj4

Cada thread deberá ejecutar una función llamada ArmoHashMapEj4, que armará el hashmap.

Primero creamos un ConcurrentHashMap y luego iteramos por cada elemento de la lista.

Por cada uno de estos ítems, creamos un thread que es el que se encargará de recorrer todo el archivo de texto.

Se usan dos iteradores: itera\_ini e itera\_fin.

El itera inicio indica el próximo archivo (un thread puede agregar varios archivos),

El itera fin es la cota.

Su deber será agregar las palabras al ConcurrentHashMap.

ArmoHashMapEj4

1: struct datos\_tread\_ejercicio4\*) datos;

2: iterator\* it = datos->itera\_ini;

3: iterator itera\_fin = datos->itera\_fin

4: pthread\_mutex\_lock(datos->mutex\_itera);

5: Mientras no es itera\_fin

pthread\_mutex\_lock(datos->mutex\_itera);

arch = \*(\*it)

(\*it)++

fin

6: pthread\_mutex\_unlock(datos->mutex\_itera);

7: string arch

8: open arch

9: mientras queden palabras en arch

10: (datos->map)->addAndInc(palabra)

11: close arch

12: pthread exit(NULL);

Creamos el ConcurrentHashMap (será compartido por todos los threads)

Se crean los hilos de ejecución y se ejecuta armoHashMapEj4, en cada uno.

Como todos pueden acceder en forma concurrente, sincronizamos utilizando phtread mutex,

Garantizamos que el iterador sea modificado una vez por vez.

El thread que pudo ‘pasar’ cambia el iterador del ultimo archivo

Una vez que el thread tiene un archivo asignado, se libera el mutex de forma que otro

thread pueda ‘pasar’ y leer así otro archivo, de manera que no hay threads ociosos.

ConcurrentHashMap count words(unsigned int n, list<string>archs)

1: ConcurrentHashMap map

2: iterator itera\_ini = archs.begin

3: iterator itera\_fin = archs.end

4: pthread\_mutex\_t mutex\_itera

5: para cada i (crea threads)

args[i] = {i, &map, &itera\_ini, itera\_fin, &mutex\_itera}

pthread\_create(&threads[i], NULL, ConcurrentHashMap::ArmoHashMapEj4, (void\*) &args[i]);

n num\_threads

6: para cada i

pthread\_join(threads[i], &status)

n num\_threads

7: return map

TESTS

Se realizaron los siguientes tests

test-1: Valida…

test-2: Valida el funcionamiento de la función count\_words que recibe como parámetro el nombre del archivo

test-3: Valida el funcionamiento de la función count\_words que recibe como parámetros el número de threads y la lista de archivos.

test-4: Valida el funcionamiento de la función count\_words que recibe como parámetros la lista de archivos.