Trabajo Práctico 1: *pthreads*

Sistemas Operativos - Primer cuatrimestre de 2018

Fecha límite de entrega: Viernes 20 de abril a las 23:59

1. Introducción

Un `ConcurrentHashMap` es una tabla de *hash* abierta, lo que significa que al haber una colisión se genera una lista enlazada dentro del *bucket*, que tiene la particularidad de estar preparada para ser utilizada por *threads* concurrentes. A su vez, su interfaz de uso es la de un `Map`, es decir, un diccionario. En este caso la simplicaremos teniendo únicamente claves que sean *strings* y valores que sean *enteros*.

Se deberá implementar un `ConcurrentHashMap` en `C++` que cumpla con la siguiente interfaz y condiciones.

- `ConcurrentHashMap()`: Constructor. Crea la tabla. La misma tendrá 26 entradas (una por cada letra del abecedario[1]). Cada entrada consta de una lista de pares $(string, entero)$. La función de hash será la primer letra del *string*.
- `void addAndInc(string key)`: Si *key* existe, incrementa su valor, si no existe, crea el par $(key, 1)$. Se debe garantizar que sólo haya contención en caso de colisión de *hash*. Esto es, deberá haber locking a nivel de cada elemento del `array`.
- `bool member(string key)`: `true` si y solo si el par (key, x) pertenece al *hash map* para algún x. Esta operación deberá ser *wait-free*.
- `pair<string, unsigned int>maximum(unsigned int nt)`: devuelve el par (k, m) tal que k es la clave con máxima cantidad de apariciones y m es ese valor. No puede ser concurrente con `addAndInc`, sí con `member`, y tiene que ser implementada con concurrencia interna. El parámetro *nt* indica la cantidad de *threads* a utilizar. Los *threads* procesarán una fila del `array`. Si no tienen filas por procesar terminarán su ejecución.

Los strings utilizados como clave solo constaran de los caracteres en el rango `a-z` (es decir, no habrá string con mayúsculas, números o signos de puntuación).

2. Ejercicios

Implementar las siguientes funciones que utilizarán un `ConcurrentHashMap`, pero no serán parte de la clase (salvo el primer punto).

1. Completar la implementación de `push_front` de la lista atómica provista por la cátedra. Utilizando esa lista, implementar la clase `ConcurrentHashMap` con las especificaciones anteriormente mencionadas.

[1] http://www.aal.edu.ar/?q=node/30

2. 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.

3. 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.

4. 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 aun hay archivos por procesar.

5. Implementar la función `pair<string, unsigned int>maximum(unsigned int p_archivos, unsigned int p_maximos, list<string>archs)`, que tome como parámetro la cantidad de *threads* a utilizar para leer los archivos (*p_archivos*), la cantidad de *threads* a utilizar para calcular máximos (*p_maximos*) y la lista de archivos a procesar (*archs*). No podrán utilizar las versiones concurrentes de la función `count_words`.

6. Realice la misma funcionalidad utilizando las versiones concurrentes de `count_words` y compare resultados.[2]

Nota: **no** puede asumir que definir un valor de un puntero (por ejemplo, `tabla[k].start = newnode`) es atómico.

3. Entregable

La entrega se realizará de manera electrónica a través del formulario que pueden encontrar en la siguiente URL: `https://goo.gl/forms/r9gE6tUChBUu2Y6z1`.

En el formulario, deberán completar los datos de cada uno de los integrantes del grupo, y subir un archivo comprimido que deberá contener **únicamente**:

- El documento del informe (en **PDF**). Deberán mostrar adecuadamente, con los instrumentos que cree necesarios (gráficos, figuras, etc), las pruebas realizadas.
- El código fuente debidamente documentado **completo y con Makefiles** de los distintos algoritmos (**NO** incluir código compilado).
- La cátedra proveerá archivos de pruebas. Se deberá, para los distintos puntos mencionados, que realicen tests mostrando la correcta implementación.
- Las modificaciones al *Makefile* para correr los test agregados.

La implementación que realicen deberá estar libre de condiciones de carrera y presentar la funcionalidad descripta anteriormente. Además, ningún proceso deberá escribir un resultado ya resuelto por otro *thread*.

El informe deberá ser de caracter **breve**. Se deberá incluir el pseudocódigo de los algoritmos, las referencias que justiquen la implementación desarrollada y la explicación de los tests desarrollados.

Al recibir correctamente una entrega, el sistema enviará un mail de confirmación a todos los integrantes del grupo. Si es necesario, podrán realizar múltiples envíos, en cuyo caso el corrector solo tendrá en cuenta el último envío que se haya realizado antes de la fecha límite de entrega.

[2] Utilice la función `clock_gettime` de la biblioteca `time.h` y linkear utilizando `-lrt`