vProgramación de Sistemas

Proyecto Final

Introducción

En este proyecto, implementarán un key-value store, para almacenamiento y búsqueda rápida de valores. En este proyecto usarán lo aprendido en E/S, sockets, hilos y sincronización.

Descripción

En este proyecto implementarán un key-value store (KV store), que permite elementos en memoria, asociados a claves. Los key-value stores son muy usados en aplicaciones distribuidas, ya que permiten rápido acceso a datos de mucho uso.

El key-value store es básicamente es **una hashtable** (https://goo.gl/1kC3aH). Un hash table nos permite rápidas operaciones de búsqueda, inserción y borrado. Esta consiste en un arreglo de **buckets,** donde se insertarán los elementos (ver figura 1). Para insertar un elemento, se pasa el elemento y una clave (en nuestro caso la palabra), la cual pasa por una **función de hashing, que convierte la clave en un numero entero**. Este número es el índice donde se insertará el elemento. Ver el grafico de abajo:

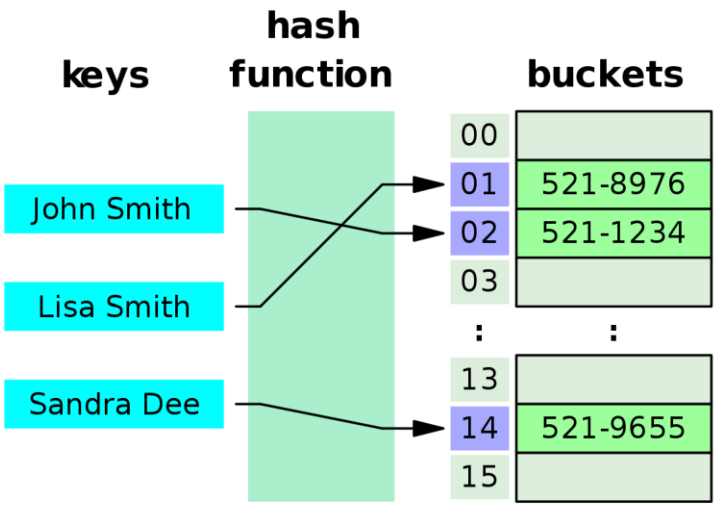

Figura 1. Hash Table

Es posible que varios elementos caigan dentro de un mismo bucket, **en cuyo caso el objeto viejo será reemplazado por el nuevo**. Ustedes implementarán el KV store en C. Representaremos los objetos con las siguientes estructuras:

```
typedef struct kvTDA{
    char *clave;                    //clave
    char *valor;                    //el objeto (puntero)
} kvObjeto;
```

```
typedef struct kvStoreTDA{
    char* id;              //Id de la hashtable
    int elementos;        //Numero de elemento en la hashtable
    int numeroBuckets;    //Numero de buckets
    kvObjeto **buckets;     //Arreglo de buckets (arreglo de punteros a
                          //objetos)
} kvStore;
```

Noten que la estructura `kvObjeto` tiene un puntero de tipo **char***: aquí es donde vamos a poner el objeto asociado a la clave.

Para saber en que bucket insertar el par clave-valor, pasamos la clave por una función de hashing. La función de hashing que van a usar es la siguiente:

```
unsigned long hash(unsigned char *str)
{
    unsigned long hash = 5381;
    int c;

    while (c = *str++)
        hash = ((hash << 5) + hash) + c; /* hash * 33 + c */

    return hash;
}
```

Para calcular el bucket, al resultado de la función hay que hacerle módulo con la cantidad de buckets:

abs(resultado_hash) % NUMERO_BUCKETS

Conectividad

El KV store será un servidor, que escuchará solicitudes para insertar, obtener remover pares clave-valor. El formato de las solicitudes el el siguiente (son strings):

1. PUT, kv_id,clave,valor
    a. Retorna OK
    b. Retorna a ERROR,causa
2. GET, kv_id,clave
    a. Retorna OK,valor
    b. Retorna ERROR,causa
3. REMOVE,kv_id,clave
    a. Retorna OK
    b. Retorna ERROR,causa
4. DELETE,kv_id
    a. Retorna OK
    b. Retorna ERROR,CAUSA

Donde:

1. kv_id es el id del KV store donde queremos hacer la operación
2. clave es la clave de clave-valor
3. valor es el valor de clave-velor
4. causa es un string con descricipción del error.

Condiciones de funcionamiento:

1. Si se hace un PUT a un kv_store que no existe, se debe crear automáticamente
2. Si se hace GET, REMOVE, DELETE a un kv_store que no existe, retornar error.
3. El valor devuelto al hacer GET son los bytes del objeto en kvObjeto.
4. El servidor puede tener activos varios KV stores al mismo tiempo
5. Al eliminar un KV store, debe liberar toda la memoria creada.

El KV store debe se **multihilo**. Es decir, debe poder manejar solicitudes concurrentes. Ya que puede haber varios hilos insertando, obteniendo o removiendo elementos al mismo tiempo en un mismo KV store, Ud. **tendrá que sincronizar el acceso al KV store donde se realice la operación.**

SU IMPLEMENTACIÓN SE CALIFICARÁ EN PARTE BASADO EN EL RENDIMIENTO DE SU IMPLEMENTACIÓN. Por lo tanto, piense formas en que puede sincronizar su hashtable (TIP: trate de no bloquear todo la hashtable si dos o más operaciones no afectan al mismo bucket). Su programa será probado insertando/obteniendo y removiendo varios gigabytes de datos, en una máquina de varios CPUs.

El programa de **kvstore** será un servidor que estará escuchando solicitudes en la red y se lo ejecutará de la siguiente manera:

./kvstore ip puerto buckets

Estos son:

1. ip es la dirección donde correrá el key-value store.
2. puerto es el puerto donde se escucharán las solicitudes.
3. Buckets es el número de buckets que tendrá el KV store

Entregable y Calificación

El proyecto es grupal. La fecha de entrega es el miércoles 23 de Agosto de 2019, 22:00. El entregable es el repositorio de Git del proyecto. El repositorio de grupo debe llamarse **proyecto_final.**

Ud debe dividir su proyecto en carpetas listadas a continuación:

1. obj
2. src
3. bin
4. include
5. Makefile

Para la calificación, se muestran algunas indicaciones generales:

1. Programas sin Makefile, makefile que no genera el programa, o compilado sin la bandera **-Wall** tienen **0 automático**.
2. Por cada **warning** se deducirá 5 puntos de la nota.
3. Entregas de archivos .zip/.rar/.7z/.tar.gz tendrán 0 automático
4. Programas con segmentation fault se calificarán a discreción del profesor.
5. Si el programa no está separado por carpetas, se calificarán sobre 80.
6. Si el programa está escrito en un solo archivo, se calificará sobre 50.
7. Si el nombre del ejecutable no es **kvstore**, se calificará sobre 90.