Design Document: Asg2

Jinghao Shen
CruzID: jshen30

1 Goals

The goal for Assignment 2 is to modify your single-threaded RPC server from Assignment 1 to provide multi-threading and to provide a simple key-value store for mathematical operations. It must still provide all of the original functionality from Assignment 1, and will support the following additional features:

1. Multiple threads, one per client, up to 16 simultaneous threads serving clients.
2. Support for variables in math operations. This requires that values be stored in a key-value store.
3. Sharing of the key-value store across all server threads, and synchronization between the server threads for reading and writing key-value pairs.

2 Design

Making a hash table for items with a fixed-size char array for the name and a 64-bit signed integer for the value. The hash table has functions insert, replacement, delete, and lookup.
To store the key-value

```
struct HT_item{
        char* name;
        int64_t value;
        HT_item* next;
}

struct hashTable{
        size_t size;
        HT_item** array;
        count;
}

import DJBHash

createItem(key, value)
        Name = key
        Value = value
        Next = null
        Return item

createTable(size)
        Table = array[size]
        return hashTable
```

```
freeTable(hashTable*)
        for(0 to size)
                if(item != null)
                        table[index]->null
        free(array)
        free(table)

Insert(hashTable*, key, value){
        newItem = createItem
        Index = hashFunction(key);
        if (array[index] == NULL){
                if(count > size)
                        Table full return error
                insert newItem to array[index]
                count++;
        }else{
                if(key == array[index] -> key){
                        update value;
                Else
                        curr = array[index]
                        while(curr -> next != null)
                                curr = curr-> next;
                        curr-> next = newitem;

replacement(hashTable*, key, value)
        index = hashFunction(key);
        if(no such key)
                return
        else
                if(array[index]-> next == null && key == key)
                        array[index]-> value = value
                else
                        while(go through linked list){
                                if(key == key){
                                        update value
                                }
                        }
        }
        return

delete(hashTable*, key)
        index = hashFunction(key);
        if(no such key)
```

```
                return
        else
                if(array[index]-> next == null && key == key)
                        Array[index] = null
                        count--;
                else
                        if(array[index]-> key == key){
                                remove array[index]
                                set array[index]->next to array[index]
                                count--;
                                return

                        while(go through linked list){
                                if(key == key){
                                        Remove from chain
                                }
                        }
        }
        return null

lookup(hashTable*, key)
        index = hashFunction(key);
        Item = array[index]
        while(item != Null){
                if(item->key = key)
                        return item->value
                if(item->next = null){
                        return null
                item = item->next
        return null

dump(file)
        Open file
        for(i=0 to size){
                if(array[i] != null){
                        dprintf("varname=value\n")
                        if(array[i]->next != null)
                                while(go through linked list)
                                        dprintf("varname=value\n")
                                }
                        }
                }
        }
        close(file)
```

```
load(file)
    Open file
    while(read)
        Extract variable name
        Extract value
        insert(table, name, value)
```

For Math functions:

```
function(buffer[], operator, ifError){
    if(del){
        Read 1 operand
    }
    Switch
        Case 0x10
            A as operand
        Case 0x20
            B as operand
        Case 0x30
            A as operand
            B as operand
        Case 0x40
            A as operand
            res as operand
        Case 0x50
            A as operand
            res as operand
        Case 0x60
            B as operand
            res as operand
        Case 0x70
            A as operand
            B as operand
            res as operand
        Default
            a = read 8 bytes from the buffer
            b = read another 8 bytes from the buffer
    }


    if(overflow)
        ifError = 22;
    else
```

```
                ifError = 0;
        if(there is a res as operand){
                insert(ht, key, value);
        }
        return a+b or a-b or a*b or a/b or a%b
```

To have multithread:

```
    Create threads[N] where N taken from argv
    Create mainMutex
        Initialize threads n times
        pthread_create
    Start the connection
    while(true){
        Cl = accept()
        while(available thread == 0){
            wait(mainMutex);
        }
        threads[i].cl = cl;
        signal(mainMutex);
    }
```