Writeup for P2 EXP2B

Christian F. Jung

Unfortunately, I was not able to completely finish the homework in time. I was struggling to wrap my head around some of the C syntax and was unable to get past Version 1 of the implementation plan. I believe that I was limited by my C skills and not my understanding of the core concepts. In this document, I will discuss what I did and what I would do to implement V2 and V3 in hopes of demonstrating my knowledge for partial credit.

What I did

When starting this homework, I began my designing a program architecture that would allow me the structure needed to accomplish the task.

image-20210408154114779

I would start by initializing what I called a "Master Hash Table" which was really an array of `N` length. Each entry in this array would be a "slave hash table" which would contain it's appropriate entries. To determine how which slave table that a new entry would go to, I would do `key % N`. In my implementation, I kept all my key and values as strings and for fun, I made my tables a grocery list. *(My family has a lot of big eaters so we need to have many threads for our list)*.

Below is an abbreviated version of the `myhash.c` function which allowed me to to instantiate the master and slave tables. One thing that I struggled with was being able to pass a GHashTable object as a parameter to a function so instead I have several repeated lines in main. *(I've kept these attempts commented out in the myhash.c in tar file)*

```
#include <glib.h>
#include <stdio.h>

int getI(int N, int  key){
return key % N;
}



int main() {

  int N = 5;
  GHashTable * m[N];

    for (int i = 0; i < N ; ++i)
  {
```

```
    printf(" creating %dth slave \n", i);

    m[i]= g_hash_table_new(g_str_hash, g_str_equal);

  }



  char * key = "0";
  char * value = "apples";
  int k = getI(N, atoi(key) );
  g_hash_table_insert(m[k],key,value);


  key = "1";
  value = "pears";
  k = getI(N, atoi(key) );
  g_hash_table_insert(m[k],key,value);



  key = "20";
  value = "pizza";
  k = getI(N, atoi(key) );
  g_hash_table_insert(m[k],key,value);



   for (int i = 0; i < N ; ++i)
  {

    printf(" loading %dth slave:", i);
    printf("There are %d keys in the hash table\n",
        g_hash_table_size(m[i]));

  }



    return 0;
}
```

What I would do next

Following my architecture diagram *(see above)*, I would implement a locking method on each of the tables. I'd likely implement this by having another array of `N` length and storing boolean values to indicate if a slave table is locked.

Then, I'd run this file multiple times with varying N and numbers of entries in the table. I'd then modify my makefile to time these results and save to a log file. In addition, I'd pass the results to the csv version of vtune.