Daniel Cender

CLC Assignment Chapter 4

May 19, 2019

Describe How Hash Function Works

The hash function that is utilized in the case study of Chapter 10 in the textbook is found below.

The function takes in a string of characters and uses an operation of bitwise shifts and “OR”ing the input bytes so that a unique output is defined for a given string, but that is also a valid position in the storage bucket for the hash string to be stored.

The function uses 3 “for” loops:

1. Iterates through all characters in the string, until it finds the first character that isn’t a whitespace. It then sets that as the length of the string, to be utilized in the other operations.
2. Iterates as many times as the length of the input, minus trailing whitespace.
3. The pack is added to with an “OR” operation using the current character in the string. Then the pack is shifted 8 bits to the left, prepping for the next iteration of loop 2.

A final XOR is performed on the pack to ensure the result is 8 bytes in length. Then, a traditional modulo TSize hash is used to generate the output.

Since these bitwise operators are new to me as of this weekend, I am still struggling to understand how they work fully in this function. I understand, however, that the OR and XOR operations are crucial here to guaranteeing a successful hash. The OR (“|”) operator returns a 1 bit if either of the bytes compared is a 1 at that position. The XOR (“^”) operator returns a 1 if either of the bytes compared is a 1 at that position, but returns a 0 if both of them are 1’s. That is why it is denoted the “exclusive” OR.

There is actually an error in this function as I modified it and ran it in CLion. “xor” is now denoted as a reserved word for C++, so it is invalid as a variable name, as it is used here.

This function is used in the File::insertion(char line[]) method. On the first line of the function, hash() is called on the string parameter passed into insertion(). Then, the address in storage is tested for pre-existing data or used to store the hash.

**unsigned long** File::hash(**char** \*s) {  
 **unsigned long xor** = 0, pack;  
 **int** i, j, slength; *// exclude trailing blanks;* **for** (slength = strlen(s); isspace(s[slength-1]); slength--);  
 **for** (i = 0; i < slength; ) {  
 **for** (pack = 0, j = 0; ; j++, i++) {  
 pack |= (**unsigned long**) s[i]; *// include s[i] in the* **if** (j == 3 || i == slength - 1) { *// rightmost byte of pack;* i++;  
 **break**;  
 }  
 pack <<= 8;  
 } *// xor at one time 8 bytes from s;* **xor** ^= pack; *// last iteration may put less*} *// than 8 bytes in pack;* **return** (**xor** % tableSize) \* bucketSize \* recordLen;  
}*// return byte position of home bucket for s;*