CS 211: Computer Architecture, Fall 2016
Programming Assignment 2: Count Unique Addresses

Due: October 19, 2016 at 11:59:59 pm

1 Introduction

This assignment is designed to use a hash table to count the number of unique addresses accessed by a program. You need to implement a program called `count`. The input of the `count` program is a trace consisting of 64-bit addresses and you are required to print out the number of unique address in the trace.

2 Problem specification: Use a hash table to count the number of unique addresses

2.1 Hash table with chaining

In this assignment, you will implement a hash table with chaining. An important part of a hash table is collision resolution. In this assignment, we want you to use chaining, which is different from Assignment 1. Figure 1 shows an example of using separate chaining in hash table. More information about chaining with linked list can be found at Wikipedia, `http://en.wikipedia.org/wiki/Hash_table#Separate_chaining_with_linked_lists`.

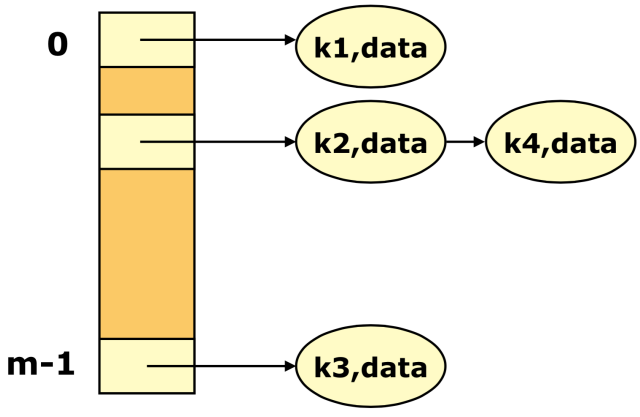

Figure 1: Example of a hash table with separate chaining. Use a linked list to store collisions.

2.2 Count number of unique addresses

Input format: This program takes a file name as argument from the command line. The file can have from a few lines to millions of lines. Each line contains an address in hexadecimal format, i.e. `0x7f1a91026b00`, generated by `pintool` (`http://pintool.org`). Each address is represented as a 64-bit hexadecimal number.

Output format: Your program should print the number of unique addresses in the file. There should be no leading or trailing white spaces in the output. Your program should print "error" (and nothing else) if the file does not exist.

Example Execution:

Lets assume we have 3 text files with the following contents. "file1.txt" is empty and,
file2.txt:
0x7f1a9804ae19
0x7f1a9804ae1c
0x7f1a9804ae1c
0x7f1a9804ae1c

file3.txt:
0x7f1a9804ae19
0x7f1a9804ae1c
0x7f1a9804ae1c
0x7f1a9804ae19
0x7f1a9804ae16
0x7f1a9814ae1c

./count file1.txt
0

./count file2.txt
2

./count file3.txt
4

./count file4.txt
error

2.3 Scalability

In this assignment, we will test your program with millions of lines of addresses. You can initialize the hash table with 1000 entries. The largest test case contains 5 million lines with about 15,000 unique addresses. In this case, the average number of nodes in each linked list is 15.

3 Submission

You have to e-submit the assignment using Sakai. Your submission should be a tar file named `pa2.tar`. To create this file, put everything that you are submitting into a directory (folder) named `pa2`. Then, `cd` into the directory containing `pa2` (that is, `pa2`'s parent directory) and run the following command:

```
tar cvf pa2.tar pa2
```

To check that you have correctly created the tar file, you should copy it (`pa2.tar`) into an empty directory and run the following command:

```
tar xvf pa2.tar
```

This should create a directory named `pa2` in the (previously) empty directory.

The `pa2` directory contain folder `count`. The `count` folder must contain c source files, header files and a Makefile. For example, after typing make, your program must generate an executable file called `count`.

- `Makefile`: there should be at least two rules in your Makefile:
  1. `count`: build your `count` executable.
  2. `clean`: prepare for rebuilding from scratch.
- source code: all source code files necessary for building `count`.

4 Grading Guidelines

We will build a binary using the Makefile and source code that you submitted, and then test the binary for correct functionality against a set of inputs. Thus:

- You should make sure that we can build your program by just running `make`.
- You should test your code as thoroughly as you can. *In particular, your code should be adept at handling exceptional cases.* For example, programs should *not* crash if the argument is not a proper number or file does not exist.
- Your program should produce the output following the example format shown in previous sections. Any variation in the output format can result **up to 100% penalty**. There should be no additional information, e.g. debugging messages.

Be careful to follow all instructions. If something doesn't seem right, ask.