CS 214: Systems Programming, Fall 2012
Programming Assignment 5: Caching

1 Introduction

In the last two assignments, you built a search tool by reading and storing a complete index in memory. When building real systems, you will often run into situations where your data set is too large to fit into physical memory. For example, if your tool is actually indexing all of your files (or, perhaps a significant part of the Web). Thus, in this assignment, you will modify your search tool to cache only a part of the index in memory at any point in time.

You should use the single-thread search tool you built for Programming Assignment 4.

2 Implementation

Extend your search tool to support the following invocation interface:

```
search [-m <memory size>] [-q <max queue size>] <inverted-index file name>
```

Your search tool should be functionally equivalent to the one implemented for the last assignment. If the optional -m flag is present, however, then your tool must use *no more* than the specified amount of memory to cache parts of the inverted index. For example, the command:

```
search -m 512MB myIndex
```

would mean that your search tool should use the inverted index stored in file `myIndex`. However, instead of reading the entire file into memory, you may use at most 512MB of memory to store parts of `myIndex`. Thus, if the user is querying for files containing the words "dog" or "cat", then your tool might read the inverted index lists for "dog" and "cat" into memory. Subsequently, if the user search for files containing "dog" and "bird", your tool might need to discard the list for "cat" before reading in the list for "bird" to keep within the specified memory constraint. You may violate the memory constraint if it is smaller than a single list.

Your tool should handle units of KB, MB, and GB for the memory size argument. It must keep careful accounting of the amount of memory currently being used to cache parts of the index file (and, of course, which parts of the index file are being cached). It has to define a caching policy; that is, what to evict from memory when a new query requires reading additional information from the index file but you have already reached the memory limit previously. Note that you do not have to worry about memory usage of other parts of your program; only the memory used to hold parts of the index file.

You will need to extend your on-disk format for storing an inverted index to make it efficient for random access. For example, if you want to read the inverted index list corresponding to the term

"zebra", it would be very inefficient to scan through the entire index file to look for the term "zebra" and its corresponding list. Remember, this index is too large to fit into memory!

You must use valgrind to ensure that your program does not have any memory leaks.

New requirement: You must use either valgrind or gprof to profile and optimize your search tool (but not the indexer). That is, you should find the functions are taking the most time and optimize them to improve performance. You should optimize at least one logical component of your program that significantly affect the speed of your search tool.

3 What to turn in

A tarred gzipped file name `pa6.tgz` that contains a directory called `pa6` containing the following:

- All the .h and .c files necessary to make your *index* and *search* programs. If you are using our indexer, you need to say it explicitly in the readme file.
- A `makefile` used to compile and produce `index` and `search`. It must have a target `clean` to prepare for a fresh compilation of everything.
- A file called `testplan.txt` that contains a test plan for your indexer and search tools. You should include the example files/directories that you index to test your search tool, but keep these from being too large, please. (We might test your program with a very large data set though so dont skip testing your program for scalability. In your test plan, you should discuss the larger scale testing and the results, but you can skip including the data set).
- A `readme.pdf` file that describes the design of your indexer and search tool. The writeup should also discuss any complaints from valgrind for memory usage. Finally, the writeup should discuss the optimizations that you did, and present measured execution times to show how much improvement you were able to achieve.

As always, your grade will be based on:

- Correctness (how well your code is working),
- Quality of your design (did you use reasonable algorithms),
- Quality of your code (how well written your code is, including modularity and comments),
- Efficiency (of your implementation), and
- Testing thoroughness (quality of your test cases).