Lab 10: Code Profiling

Nathan Jarus

April 4, 2016

Introduction

This lab will give you experience using both CPU and memory profilers. Please make an answers file and commit it to the repository along with your code fixes.

Problem 1: Massif

Remember ye olde filter.cpp? It's back again!

- Build `prob1.cpp` and run it through massif (`valgrind --tool=massif --time-unit=B ./prob1 < story.txt`). How much memory does the program allocate in total (cumulative)? What is the maximum amount of memory allocated at one time?
- Open up `prob1.cpp`. Do you really need to store every line of the input if you're just printing it out? Fix the code so that you only keep the current line of input.
- Run the fixed code through massif again. Now, what is the total cumulative memory allocation and peak memory allocation?

Problem 2: gprof

This problem's code appends one random number between 1 and 1000 to a vector and prints the average of the vector. Hopefully, the average will converge to around 500.

- Build `prob2.cpp` for gprof and run it, then look at gprof's output. What functions are called frequently? (It may help to call `gprof -A`.)
- Look at the average function. Why is the vector's copy constructor being called? Fix the code to not make an unnecessary copy.
- Run the code through gprof again. Did your fix work?

Problem 3: callgrind

This problem's code calculates the length of input lines and prints a running average.

- Build `prob3.cpp` and run it through callgrind (see Prob 1.1). Run `callgrind_annotate --auto=yes callgrind.out.NNNN`. What lines in `main()` consume a lot of instructions? How many instructions are spent calculating the average? (lines 17-27)
- Look at the source code. Do we really need to re-count the length of each line each time we calculate the average? (Hint: no.) Fix the code to maintain a running total.
- Re-build and run the code through callgrind. How many instructions are spent calculating the average now?