Princeton University
COS 217: Introduction to Programming Systems
Execution Profiler: Development Stages

Stage 1: A Simplified instrument Program

Design a simplified version of the instrument program. The simplified instrument program should add code at the beginning of each assembly language function as appropriate. The simplified instrument program should also add code at the end of the assembly language program that creates function name strings in the rodata section.

Testing:

Design a suite of testing C programs. Use "gcc –S" to create assembly language programs from those C programs. Instrument each of the assembly language programs. Examine the results.

Stage 2: A Simplified __count() Function

Design a simplified __count() function (and subordinate functions) that uses a SymTable object to accumulate function call counts, and that writes those counts to a "stats" file immediately before process exit.

Testing:

Enhance your suite of testing C programs as appropriate. Compile, instrument, assemble, and link those programs. Run those programs. Examine the results in the stats file. The function call counts in the stats file should be identical to those reported by gprof.

Stage 3: The Complete instrument Program

Enhance your simplified instrument program from Stage 1, thus designing the complete instrument program. Specifically, enhance your simplified instrument program so it also adds code at the end of each function as appropriate, and so it adds code to the end of the assembly language program that creates a "function name / function start address / function end address" table in the rodata section.

Testing:

Enhance your suite of testing C programs as appropriate. Use "gcc –S" to create assembly language programs from those C programs. Instrument each of the assembly language programs. Examine the results.

Stage 4: The Complete __count() function

Enhance your simplified __count() function (and subordinate functions) from Stage 2, thus designing the complete __count() function. Specifically, enhance your simplified __count() function so it uses signals and interval timers to accumulate function signal counts, and writes those counts to a "stats" file immediately before process exit.

Testing:

Enhance your suite of testing C programs as appropriate. Compile, instrument, assemble, and link those programs. Run those programs. Examine the results in the stats file. The function call counts in the stats file should be identical to those reported by gprof. The signal counts in the stats file should be proportionate to the times reported by gprof.

Stage 5: Testing Strategy

Describe your strategy for testing the complete execution profiler in your readme file.