

Dynamic Analysis

CSCI 3308 – Lab11

Program Analysis

Program analysis is a technique that reasons about the run-time behavior of the program

- Static program analysis – reasoning is done statically, before program execution

- Dynamic program analysis – reasoning is done dynamically, during program execution

GDB - GNU Debugger

GDB is the standard debugger for the GNU operating system. However, its use is not strictly limited to the GNU operating system; it is a portable debugger that runs on many Unix-like systems and works for many programming languages, including Ada, C, C++, Objective-C, Free Pascal, Fortran, Java and partially others.

GDB - Examples of commands

<code>gdb ./<executable></code>	debug "program" (from the shell)
<code>r</code>	run the loaded program with the parameters
<code>bt</code>	backtrace (in case the program crashed)
<code>b</code>	Set Breakpoint
<code>n</code>	Execute the next line
<code>p</code>	Print the value of variable
Info locals	Prints values of variables in scope
<code>q</code>	Quit GDB

Valgrind

Valgrind is an instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail. You can also use Valgrind to build new tools.

valgrind --tool=memcheck --leak-check=full ./<executable>

```
Terminal
File Edit View Terminal Tabs Help

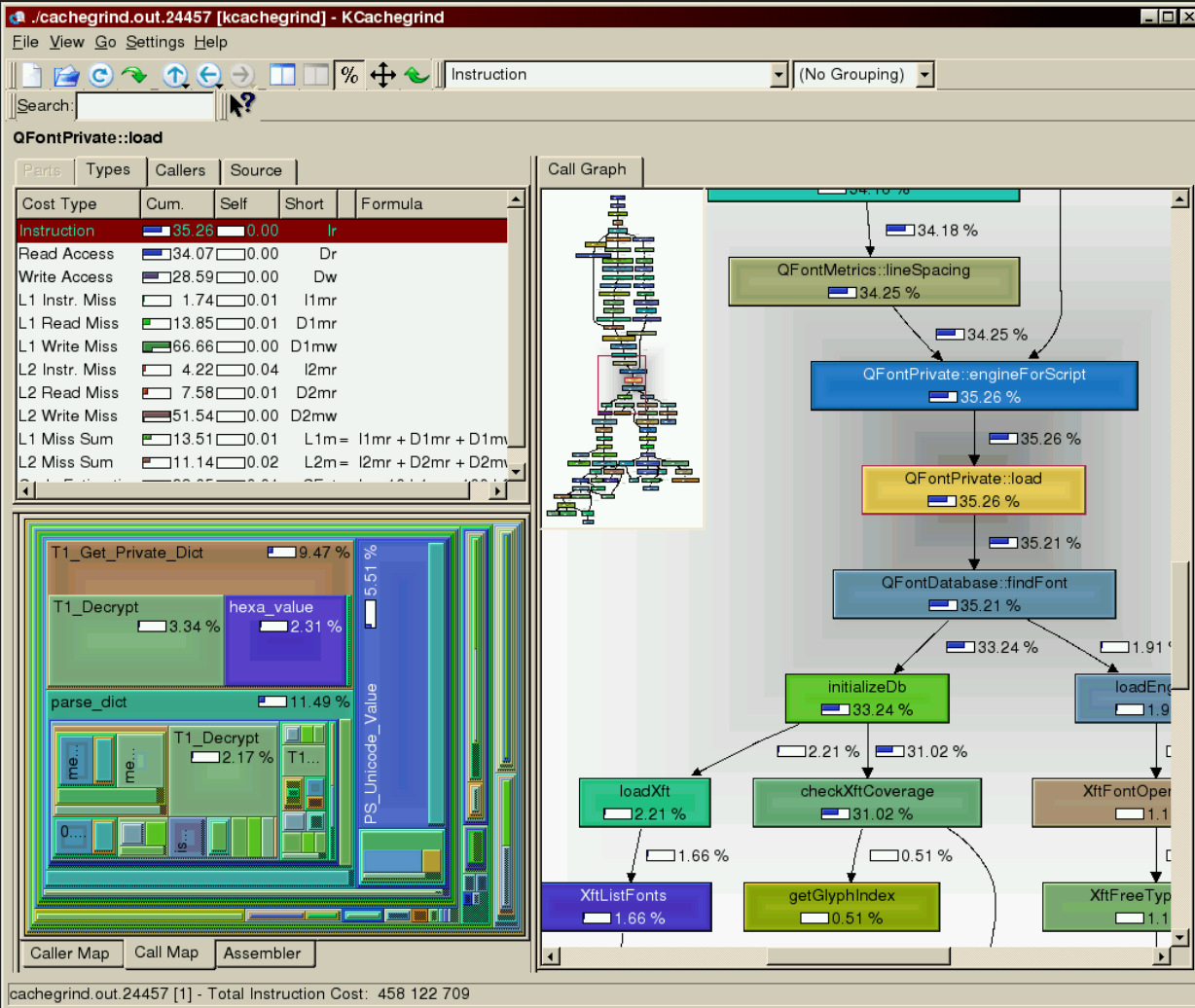
[pwells2@newcell ~/junk]$ valgrind --leak-check=full ./memleak
==16547== Memcheck, a memory error detector
==16547== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==16547== Using Valgrind-3.6.1 and LibVEX; rerun with -h for copyright info
==16547== Command: ./memleak
==16547==
==16547== Invalid write of size 4
==16547==    at 0x400589: main (mem_leak.c:32)
==16547== Address 0x4c26068 is 0 bytes after a block of size 40 alloc'd
==16547==    at 0x4A0646F: malloc (vg_replace_malloc.c:236)
==16547==    by 0x400505: main (mem_leak.c:17)
==16547==
==16547== Invalid read of size 4
==16547==    at 0x400598: main (mem_leak.c:33)
==16547== Address 0x4c26068 is 0 bytes after a block of size 40 alloc'd
==16547==    at 0x4A0646F: malloc (vg_replace_malloc.c:236)
==16547==    by 0x400505: main (mem_leak.c:17)
==16547==
==16547==
==16547== HEAP SUMMARY:
==16547==    in use at exit: 410 bytes in 8 blocks
==16547== total heap usage: 11 allocs, 3 frees, 590 bytes allocated
==16547==
==16547== 40 bytes in 1 blocks are definitely lost in loss record 1 of 2
==16547==    at 0x4A0646F: malloc (vg_replace_malloc.c:236)
==16547==    by 0x400505: main (mem_leak.c:17)
==16547==
==16547== 370 bytes in 7 blocks are definitely lost in loss record 2 of 2
==16547==    at 0x4A0646F: malloc (vg_replace_malloc.c:236)
==16547==    by 0x400528: main (mem_leak.c:23)
==16547==
==16547== LEAK SUMMARY:
==16547==    definitely lost: 410 bytes in 8 blocks
==16547==    indirectly lost: 0 bytes in 0 blocks
==16547==    possibly lost: 0 bytes in 0 blocks
==16547==    still reachable: 0 bytes in 0 blocks
==16547==    suppressed: 0 bytes in 0 blocks
==16547==
==16547== For counts of detected and suppressed errors, rerun with: -v
==16547== ERROR SUMMARY: 38 errors from 4 contexts (suppressed: 4 from 4)
[pwells2@newcell ~/junk]$
```

Profiling

In software engineering, profiling ("program profiling", "software profiling") is a form of dynamic program analysis that measures, for example, the space (memory) or time complexity of a program, the usage of particular instructions, or the frequency and duration of function calls. Most commonly, profiling information serves to aid program optimization.

Callgrind and KCACHEGRIND

Callgrind uses runtime instrumentation via the Valgrind framework for its cache simulation and call-graph generation. This way, even shared libraries and dynamically opened plugins can be profiled. The data files generated by Callgrind can be loaded into **KCachegrind** for browsing the performance results. But there is also a command line tool in the package to get ASCII reports from data files without the need to use KCachegrind.



valgrind --tool=callgrind
./<executable>

kcachegrind callgrind.out.8117

Optional:

<https://www.youtube.com/watch?v=y5JmQItfFck>

<https://www.youtube.com/watch?v=fvTsFjDuag8>