Profiling

- Was ist ein Profiler? (Theorie)
  - Invasives Profiling
  - Nichtinvasives Profiling
- Profiling in der Praxis
  - gprof, gcov
  - OProfile
  - valgrind/callgrind
  - Intel® VTune™

Was ist ein Profiler?

Analyse des Laufzeitverhaltens eines Programms:

- Welche Teile (Funktionen, Anweisungen, etc.) benötigen wie viel Zeit?
- Wie oft werden Funktionen aufgerufen?
- Welche Funktionen werden von welchen Funktionen aufgerufen (Aufrufgraph)?
- Speicherverhalten (Speicherzugriffe, Speicherlöcher, Cache, ...)

Invasives Profiling

- Instrumentierung des Codes (Einfügen von Aufrufen an Überwachungsfunktionen)
- Vorteile:
  - Potentiell beliebig genau
  - Aufrufgraph
- Nachteile:
  - Sehr hoher Overhead
  - ungeeignet für lang laufende (Server-)Prozesse (Ausgabe?)

Nichtinvasives Profiling

- HW Performance Counter (CPU Feature)
- Statistische Analyse (Analyse des PC und des Stacks bei Timer Interrupt)
  - Histogramm, Sampling
- Vorteile:
  - geringer Overhead (steuerbar)
  - Systemperformance messbar
- Nachteile:
  - nicht exakt
  - Möglicherweise unvollständiger Aufrufgraph

Profilformen

- “Flaches Profil” (flat profile)
  - wieviel Zeit verbringt das Programm in welcher Funktion?
- “Aufrufgraph” (call graph)
  - enthält Informationen darüber, welche Funktion von wo aus aufgerufen wurde und wie oft
- “Annotierte Quellen” (annotated source)
  - annotierte Quellen bestehen aus dem Quellcode des Programms, annotiert mit der Anzahl der Ausführungen für jede Zeile

gprof

- Statistische Analyse des PC:
  - profil() Syscall aktiviert Profiling im Kernel
  - periodisch (100/s bei Timer-Interrupt) wird PC in Histogramm eingetragen
- Aufrufgraph (invasiv):
  - zur Übersetzungszeit werden Aufrufe an mcount() an Anfang jeder Funktion gesetzt
  - mcount() ermittelt aus Stack Frame (frompc, topc) und baut internen Aufrufgraph auf
  - Binden mit C-Bibliothek, die Profiling-Info enthält, um Aufrufstatistik für Bibliotheksroutinen zu erhalten

gprof (2)

- Compilerflags:
  - “-pg”: Instrumentierung fuer Aufrufgraph und Initialisierung der Statistik
  - “-g”: Debug-Info (Abbildung Adresse auf Codezeile) für annotierte Quellen
- Anwendungsgebiete:
  - grober Überblick über Zeitverbrauch von Funktionen

gcov

- analysiert Überdeckung des Programmcodes (welche Zeile wurde wie oft ausgeführt)
- hilft bei der Erkennung von Code
  - bei dem Optimierung lohnt
  - der von Testdaten benutzt wird (ungetestete Bereiche finden)
- Compilerflags:
  - “-fprofile-arcs”: Infos über Sprünge sammeln
  - “-ftest-coverage”: Infos über Codeüberdeckung sammeln

gcov (2)

- Anwendungsgebiete:
  - zeilengenaue Analyse
  - Analyse von Verzweigungswahrscheinlichkeiten (eigentlich grundblockgenau)
- Achtung:
  - wenn mehrere Codezeilen zu einer Instruktion werden sind keine Zahlen für einzelne Zeilen verfügbar:

```
100:   12: if (a != b)
100:   13:   c = 1;
100:   14: else
100:   15:   c = 0;
```

OProfile

- Kernel-Modul und Daemon
- Profil für *gesamtes* System
- nichtinvasiv:
  - nutzt HW Performance Counter für Ereignisse
  - falls Performance Counter nicht verfügbar:
    - Timer/RTC-Modus
- Aufrufgraph im gprof-Format
- Profil für jedes Binary separat
- annotierte Quellen möglich, falls Debug-Symbole vorhanden

OProfile (2)

- Vorteile:
  - geringer Overhead
  - keine Neuübersetzung nötig
  - Messdaten werden periodisch auf Platte geschrieben
  - kann auch Profil für Interrupthandler erstellen
  - kann HW-Effekte wie Cache Misses messen
- Nachteile:
  - HW Performance Counter nur für x86 unterstützt
  - benötigt root-Rechte
  - Profile nicht 100% korrekt auf Anweisungsebene
  - kein Funktionsaufrufzähler

CPU-Unterstützung

- Performance Counter:
  - HW Register, die Ereignisse zählen, wie z.B. Cache Misses, CPU Cycles
- Einsatz:
  - Kernel-Profiling (Interrupthandler etc.)
  - erhöhte Genauigkeit bei nichtinvasivem Profiling

valgrind

- virtuelle x86 CPU
- Instrumentierung des Maschinencodes
- mehrere Werkzeugmodule:
  - *Memcheck*: erkennt Probleme im Speichermanagement, z.B. Verwendung nichtallozierter Speicherbereiche, Memory Leaks
  - *Addrcheck*: leichtgewichtige Variante von Memcheck ohne Prüfung auf uninitialisierten Speicher
  - *Cachegrind*: Simulation von L1-/L2-/D1-Caches
  - *Callgrind* (früher “calltree”)
- KCachegrind: graphisches Frontend für valgrind

valgrind (2)

- Compilerflags:
  - eigentlich keine besondere Übersetzung nötig
  - “-fno-inline”
  - “-g”, wenn man Zeileninfo möchte
- Anwendungsgebiete:
  - liefert sehr exakte Resultate
  - für laufzeitunkritische Programme
- Nachteil:
  - langsam: memcheck 25-50x Laufzeit, 12x Codegröße)

Intel® Vtune™ Performance Analyzer

- Nichtinvasive systemweite Analyse
- Eclipse GUI
- Unterstützung nur für Intel CPUs
- Visualisierung von Aufrufgraph
- Vorteile:
  - geringer Overhead
  - Integration der Werkzeuge mit GUI
- Nachteile:
  - proprietär, kommerziell, böse etc. ;-)
  - weder kostenlos noch frei (600€-5k€)
  - keine herausragenden Vorteile gegenüber OProfile

Ressourcen

- gprof:
  - man/info gprof
- gcov:
  - man/info gcov
- OProfile:
  - http://oprofile.sourceforge.net/
- valgrind/callgrind
  - http://valgrind.org/
- Intel® VTune™
  - http://www.intel.com/software/products/vtune/