# Advanced C Programming

## Profiling

Sebastian Hack
hack@cs.uni-sb.de

Christoph Weidenbach
weidenbach@mpi-inf.mpg.de

25.11.2008

# Today

## Profiling

- Invasive Profiling
- Non-Invasive Profiling

## Tools

- gprof
- gcov
- valgrind
- oprofile

## Conclusion

# What is a Profiler?

Analyse the runtime behavior of the program

- Which parts (functions, statements, . . . ) of a program take how long?
- How often are functions called?
- Which functions call which
  - Construct the dynamic call graph
- Memory consumption
  - Memory accesses
  - memory leaks
  - Cache performance

# Invasive Profiling

- Modify the program (code instrumentation)
- Insert calls to functions that record data

- Advantages:
  - Very precise
  - Theoretically at the instruction level
  - Precise call graph
- Disadvantages:
  - Potentially very high overhead
  - Depends on the instrumentation code that is inserted
  - Cannot profile already running systems (long running servers)
  - Can only profile application (not complete system)

# Non-Invasive Profiling

- Statistic sampling of the program
- Use a fixed time interval
  or Hardware performance counters (CPU feature)
  to trigger sampling events
- Record instruction pointer at each sampling event

- Advantages:
  - Small overhead
  - Hardware assisted
  - Can profile the whole system (even the kernel!)
- Disadvantages:
  - not precise ☞ "only" statistical data
  - Call Graph possibly not complete
    ☞ some functions are never sampled

# Profiles

- Flat Profile
  How much time does the program spend in which function?
- Call Graph
  Which function calls which function how often?
- Annotated Sources
  Annotate each source line with number of executions

# gprof

- Mixture of invasive and statistical profiling

## Invasive Part

- `gcc` inserts calls to a function `mcount` into prologue of each function
- Compile with `-g` and `-pg`
- `mcount` can figure out its caller ☞ we can construct the call graph
- `mcount` counts the number of invocations for each function
- Call to `mcount` is the only instrumentation
  ☞ almost as efficient as normal build
- After program is run, there is a file called `gmon.out` containing profiling data
- Evaluate contents of `gmon.out` with `gprof name-of-program`

# gprof

## Statistical Part

- Kernel samples instruction pointer (IP)
  on each timer interrupt (100/s)
- Increments a counter in a histogram of address ranges
  ☞ cannot track the exact location where timer interrupt happened
- Provides a frequency distribution over code locations
- Beware of low samplerate
- Short running programs will mostly not provide meaningful data
- Accumulation of several profile runs is possible:

```
$ ./test_program
$ mv gmon.out gmon.sum
$ ./test_program
$ gprof -s ./test_program gmon.out gmon.sum
```

# gcov

- Analyses coverage of program code
- Which line was executed how often
- Helps for finding code that
  - can profit from optimizations
  - that is not covered by test cases
- Use GCC flags
  - `-fprofile-arcs`: collect info about jumps
  - `-ftest-coverage`: collect info about code coverage
- Attention: Multiple code lines might be merged to one instruction

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

# valgrind

- JIT-compiler / translator:
  - Construct intermediate representation from x86 assembly code
  - Add instrumentation code
  - Compile back to x86
- Done while program is loaded
- Is not only a profiler!
- No compiler flags / recompilation needed
  (though `-g -fno-inline` advisable to analyse output)
- Program runtime can degrade drastically due to instrumentation code and recompilation
- can escape to debugger on certain events
  ☞ very handy when debugging memory leaks
- Disadvantage:
  - program might run an order of magnitude slower
  - program might consume an order of magnitude more memory

# valgrind

## Tools

### memcheck

- Redirects calls to `malloc` and the like
- Keeps track of all allocated memory
- Instruments references to warn about "bad" memory accesses
  - uninitialized
  - already freed
- Detects memory leaks
- Warns about jumps taken upon uninitialized values

### cachegrind

- Instruments memory accesses
- Simulates (!) a L1 and L2 cache in software
- Gives precise data about cache misses

### callgrind

- Records the call graph

**Hint**

Use `kcachegrind` for visualization

# oprofile

- Non-invasive
- Kernel module and user-space daemon
- Does not modify the program at all
- −g for debug symbols recommendable
- Sampling uses performance counters
- . . . or timer interrupt of perf. counters not available
- Profiles the whole system (also the kernel!)
- Can distill data for each binary separately
- For Windows, use Intel vTune ($$$)

# oprofile

## Performance Counter

- Set of hardware registers for a plethora of events
- Differ from processor model to another
- Very detailed events trackable. Examples:
  - L2 cache misses
  - Retired instructions
  - Outstanding bus requests
  - . . . and many more
- Basic modus operandi:
  - Kernel module tells the CPU to fire an exception after a certain number of events of a certain type have occurred
  - CPU traps into kernel
  - instruction pointer is recorded in a buffer (no histograms)

# oprofile

## Howto

- Use `opcontrol` to control the daemon/module
- `opcontrol --init` to load module and daemon
- `opcontrol -s` to start sampling
- `opcontrol -t` to stop sampling
- `opcontrol --dump` flushes the event log
- `opcontrol --list-events` shows available performance counters
- `opreport -l prog-name` gives breakdown of samples per function in `prog-name`

# Conclusion

- Many different profiling methods exist
- gprof
  - is obsolete
  - use only to get a quick impression
  - and for the call graph
  - sampling might be too imprecise
- valgrind
  - easy to use
  - no recompile
  - precise
  - good visualization (kcachegrind)
  - but large increase in runtime
- oprofile
  - much more precise than `gprof`
  - can profile exotic machine events
    if you are going for the last cycles
  - not as precise as valgrind
  - need root rights on the machine