Debugging
gdb
Compilation
A short example
Breakpoints
Watches
Execution control
Interacting with variables
Backtrace and frames
ddd
Memory error detection
valgrind
gprof
gprof
kcachegrind
Conclusion

Debugging and Profiling

October 7, 2016

Outline

Debugging and Profiling

Debugging
gdb
Compilation
A short example
Breakpoints
Watches
Execution control
Interacting with variables
Backtrace and frames
ddd

Memory error detection
valgrind

gprof
gprof
kcachegrind

Conclusion

- Debugging
  - `gdb`
  - `ddd`
- Memory error detection
  - `valgrind`
- Profiling
  - `gprof`
  - `kcachegrind`

Why debugging?

Debugging
gdb
Compilation
A short example
Breakpoints
Watches
Execution control
Interacting with variables
Backtrace and frames
ddd
Memory error detection
valgrind
gprof
gprof
kcachegrind
Conclusion

- Cheap solution:

```
std::cout << my_variable << endl;
```

- Advantages of a debugger:
  - No need to recompile
  - Global vision of the code
  - Easy to interact with the flow of the program
  - Easy to read the value of the variables

gdb - Compilation

Debugging
gdb
Compilation
A short example
Breakpoints
Watches
Execution control
Interacting with variables
Backtrace and frames
ddd
Memory error detection
valgrind
gprof
gprof
kcachegrind
Conclusion

- Compilation flag: `-g2`

A short example

Debugging
gdb
Compilation
A short example
Breakpoints
Watches
Execution control
Interacting with variables
Backtrace and frames
ddd
Memory error detection
valgrind
gprof
gprof
kcachegrind
Conclusion

`gdb PROGRAM`	Start gdb
`break 42`	Set a breakpoint
`run` [arguments]	launch the program (will stop at breakpoint)
`list`	Show surrounding code
`print x`	Display variable value
`step` `next`	Go to next line of code (w/o entering functions)

A short example

Debugging
gdb
Compilation
A short example
Breakpoints
Watches
Execution control
Interacting with variables
Backtrace and frames
ddd
Memory error detection
valgrind
gprof
gprof
kcachegrind
Conclusion

- `continue` — Resume execution till next breakpoint
- `kill` — Kill the program
- `quit (q)` ctrl-d — Exit gdb

Breakpoints

- `break (b) 42` — Set a breakpoint line 42
- `break FILE:42` — Set it line 42 in `FILE`
- `break` — Set a breakpoint at the next line
- `break +3` — Set a breakpoint 3 lines after
- `break -3` — Set a breakpoint 3 lines before

- `break FUNCTION[(arguments types)]`
- `break CLASS::FUNCTION[(arguments types)]`

  set break point at `FUNCTION`
  (if ambiguous, choice given)

- `break 2 if i>8` — Conditional break

- `tbreak` — Will stop only once

Breakpoints

Debugging
gdb
Compilation
A short example
Breakpoints
Watches
Execution control
Interacting with variables
Backtrace and frames
ddd
Memory error detection
valgrind
gprof
gprof
kcachegrind
Conclusion

- `info breakpoint` — Breakpoint management
- `disable 2`
- `delete 2`
- `ignore 2 3`

- ```
  commands 4
      print i
  end
  ```
  Commands executed when breakpoint reached

- `ctrl-c` — Pauses the execution
  ```

Watches

Debugging
gdb
Compilation
A short example
Breakpoints
Watches
Execution control
Interacting with variables
Backtrace and frames
ddd
Memory error detection
valgrind
gprof
gprof
kcachegrind
Conclusion

- `watch x` Breaks when `x` modified
- `rwatch x` read
- `awatch x` both

  (Works only when the program is running)

- `info watchpoints`
- `info breakpoints`
- `disable 2`
- Managed in the same way as breakpoints

Controlling the execution

- `step (s)` — Goes to the next line
- `next (n)` — (without entering functions)

- `step 2` — Jump several lines
- `next 3`

- `until (u)` — Runs till next line reached (useful to finish loop from its last line)
- `until 42` — Runs till line 42 reached

- `continue (c)` — Resume execution
- `finish (f)` — Finish execution of a function
- `kill (k)` — Kill program

Backward debugging

Debugging
gdb
Compilation
A short example
Breakpoints
Watches
Execution control
Interacting with variables
Backtrace and frames
ddd
Memory error detection
valgrind
gprof
gprof
kcachegrind
Conclusion

- `record` (Program must be running)
- `reverse-step (rs)`
- `reverse-next (rn)`
- `reverse-continue (rc)`
- `reverse-finish`
- `set exec-direction forward|reverse`
- `record stop`

Interacting with variables

Debugging
gdb
Compilation
A short example
Breakpoints
Watches
Execution control
Interacting with variables
Backtrace and frames
ddd
Memory error detection
valgrind
gprof
gprof
kcachegrind
Conclusion

- `print x`
- `set var x=3`
- `list`

Backtrace and frames

Debugging
gdb
Compilation
A short example
Breakpoints
Watches
Execution control
Interacting with variables
Backtrace and frames
ddd
Memory error detection
valgrind
gprof
gprof
kcachegrind
Conclusion

- `backtrace`
- `frame 1`
- `info frame`
- `info locals`
- `info args`

ddd

Debugging
gdb
Compilation
A short example
Breakpoints
Watches
Execution control
Interacting with variables
Backtrace and frames
ddd
Memory error detection
valgrind
gprof
gprof
kcachegrind
Conclusion

- `ddd PROGRAM`

Usefulness of `valgrind`

Debugging
gdb
Compilation
A short example
Breakpoints
Watches
Execution control
Interacting with variables
Backtrace and frames
ddd
Memory error detection
valgrind
gprof
gprof
kcachegrind
Conclusion

- Identification of memory issues
- Can find subtle bugs or issues, some of which will not necessarily have an immediate manifestation

valgrind

Debugging
gdb
Compilation
A short example
Breakpoints
Watches
Execution control
Interacting with variables
Backtrace and frames
ddd
Memory error detection
valgrind
gprof
gprof
kcachegrind
Conclusion

- `valgrind PROGRAM`

Profiling - gprof

Debugging and Profiling

Debugging
gdb
Compilation
A short example
Breakpoints
Watches
Execution control
Interacting with variables
Backtrace and frames
ddd
Memory error detection
valgrind
gprof
gprof
kcachegrind
Conclusion

- Compilation flag `-pg`
- Progam execution $\rightarrow$ file `gmon.out` produced
- gprof PROGRAM gmon.out

kcachegrind

Debugging
gdb
Compilation
A short example
Breakpoints
Watches
Execution control
Interacting with variables
Backtrace and frames
ddd
Memory error detection
valgrind
gprof
gprof
kcachegrind
Conclusion

- `valgrind --tool=callgrind PROGRAM [ARGS]`
- Program execution $\rightarrow$ `callgrind.out.PID`
- `kcachegrind callgrind.out.PID`

Conclusion

Debugging
gdb
Compilation
A short example
Breakpoints
Watches
Execution control
Interacting with variables
Backtrace and frames
ddd
Memory error detection
valgrind
gprof
gprof
kcachegrind
Conclusion

- **gdb**
  - More efficient than merely displaying variables
  - Possibility to interact with program flow and see variables values
- **valgrind**
  - Good practice
  - Can identity subtle issues
- **gprof / callgrind**
  - Identification of bottlenecks and places worthy of optimisation