Advanced Practical Programming for Scientists

Software Processes and Tools

Robert Gottwald, Matthias Miltenberger, Daniel Rehfeldt

Zuse Institute Berlin

June 9th, 2017
Software Processes and Tools

Agile Development and Continuous Integration

Tools for C/C++

Time Measurement
Agile Development

- avoid static/fixed development plans
- enable fast reaction to failures and problems
- esp. useful/applicable for small teams
- many agile methods
  - scrum
  - extreme programming
  - crystal clear
- basic concepts
  - pair/peer programming
  - code refactoring
  - continuous integration
  - release often
  - ...
Continuous Integration

- software engineering practice
Basic Concepts of CI

- automatic builds
- frequent (unit) tests
- dedicated build machine/server
- quick test results and analysis
- immediate feedback on broken builds (e.g. e-mail)
- requires version control (e.g. git)
Tools for CI

- Jenkins\(^1\)

  - features:
    - free, open source
    - written in Java
    - probably most popular (open source) CI tool right now
    - web interface
    - easy to deploy (java jenkins.jar)
    - easy to maintain (complete control using web interface)
    - large active community
    - many different plugins to extend the features
      (in fact, Jenkins is more of a framework that relies on its plugins)
    - different types of permission control (e.g. LDAP, or Jenkins’ own one)

\(^1\)https://jenkins.io/
Tools for CI

- hosted CI tool: Travis-CI
  - features:
    - build and test projects hosted on GitHub.com
    - free for open source projects
    - supports many programming languages
    - Berlin based company

²https://travis-ci.org/
**Next Step**

**Continuous Deployment**

- directly release build that passes all tests
- eliminates long release preparations
- users immediately get new features
- **but:** not always applicable in practice
  (users may need to integrate new versions by hand)
Software Processes and Tools

Agile Development and Continuous Integration

Tools for C/C++

Time Measurement
valgrind³

▶ Integrates different tools for analyzing programs at runtime

**Memcheck**  For detecting memory-management problems
**Callgrind**  performance profiler
**Cachegrind** cache profiler
**Massif**  heap profiler
**Helgrind**  thread debugger

³[http://valgrind.org/](http://valgrind.org/)
Static analysis

- run time checks like valgrind only find errors in the parts that are executed
- static code analysis can find errors at compile time
  - division by zero
  - integer overflows
  - uninitialized values
  - null pointer dereferences
- especially useful for hard to find bugs that appear only in some executions
- for C/C++:
  - clang static analyzer https://clang-analyzer.llvm.org/
  - cppcheck http://cppcheck.sourceforge.net/
  - coverity http://www.coverity.com/
Time for a demo!
Software Processes and Tools

Agile Development and Continuous Integration

Tools for C/C++

Time Measurement
Time Measurement

How to measure “execution time” of program that starts at time $t_0$ and ends at time $t_1$?
How to measure “execution time” of program that starts at time $t_0$ and ends at time $t_1$?

- $t_1 - t_0$? (wall-clock time or wall time)
  - Problem: what about the time spent in I/O etc.?
How to measure “execution time” of program that starts at time $t_0$ and ends at time $t_1$?

- $t_1 - t_0$? *(wall-clock time or wall time)*
  - Problem: what about the time spent in I/O etc.?
- Alternative: measure only the time spent in CPU? *(CPU time)*
  - Problem: is this time actually being spent in the program?
How to measure “execution time” of program that starts at time $t_0$ and ends at time $t_1$?

- $t_1 - t_0$? (wall-clock time or wall time)
  - Problem: what about the time spent in I/O etc.?
- Alternative: measure only the time spent in CPU? (CPU time)
  - Problem: is this time actually being spent in the program?
- Alternative: measure time CPU spent executing code in user space (user time)
CPU Time

- **CPU time**: amount of time CPU was used for processing instructions of program or of operating system. Can therefore be divided in
  - user time
  - system time (time spent in kernel space)
CPU Time

- **CPU time**: amount of time CPU was used for processing instructions of program or of operating system. Can therefore be divided in
  - **user time**
  - **system time** (time spent in kernel space)
- CPU time is measured in *clock ticks* (computers’ notion of passing of time) or seconds
- CPU time $\leq$ wall time.
CPU Time

- **CPU time**: amount of time CPU was used for processing instructions of program or of operating system. Can therefore be divided in
  - user time
  - system time (time spent in kernel space)
- CPU time is measured in **clock ticks** (computers’ notion of passing of time) or seconds
- CPU time $\leq$ wall time.
- **CPU usage** refers to CPU time as a percentage of CPU’s capacity
**CPU Time in Unix**

The `top` or `htop` command is used to get information including user time and details of all processes.

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>14653</td>
<td>bzfrehfe</td>
<td>20</td>
<td>0</td>
<td>3076624</td>
<td>1.022g</td>
<td>193540</td>
<td>S</td>
<td>4.0</td>
<td>3.3</td>
<td>64:57.84</td>
<td>Web Content</td>
</tr>
<tr>
<td>21155</td>
<td>adm_timo</td>
<td>20</td>
<td>0</td>
<td>4376940</td>
<td>217388</td>
<td>17948</td>
<td>S</td>
<td>0.7</td>
<td>0.7</td>
<td>0:29.97</td>
<td>java</td>
</tr>
<tr>
<td>809</td>
<td>mongodb</td>
<td>20</td>
<td>0</td>
<td>388320</td>
<td>52136</td>
<td>18092</td>
<td>S</td>
<td>0.3</td>
<td>0.2</td>
<td>23:55.25</td>
<td>mongod</td>
</tr>
<tr>
<td>14400</td>
<td>bzfrehfe</td>
<td>20</td>
<td>0</td>
<td>2557064</td>
<td>426296</td>
<td>151308</td>
<td>S</td>
<td>0.3</td>
<td>1.3</td>
<td>40:39.68</td>
<td>firefox</td>
</tr>
<tr>
<td>1</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>185660</td>
<td>6252</td>
<td>3944</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:03.20</td>
<td>systemd</td>
</tr>
<tr>
<td>2</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.13</td>
<td>kthreadd</td>
</tr>
<tr>
<td>3</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:02.00</td>
<td>ksoftirqd/0</td>
</tr>
<tr>
<td>7</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>3:01.94</td>
<td>rcu_sched</td>
</tr>
<tr>
<td>8</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>rcu_bh</td>
</tr>
<tr>
<td>9</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:01.58</td>
<td>migration/0</td>
</tr>
<tr>
<td>10</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:01.47</td>
<td>watchdog/0</td>
</tr>
<tr>
<td>11</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:01.46</td>
<td>watchdog/1</td>
</tr>
<tr>
<td>12</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:01.65</td>
<td>migration/1</td>
</tr>
<tr>
<td>13</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:01.41</td>
<td>ksoftirqd/1</td>
</tr>
<tr>
<td>14</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:01.29</td>
<td>watchdog/0</td>
</tr>
</tbody>
</table>
Execution Time of Your Program (Unix)

/usr/bin/time -v to measure your program. For exercise 1:

```
[08.06.17, 16:17] bzfrehfe@opt59 ~/projects/lectures/appfs17/appfs/exercisel (master)
<91> /usr/bin/time -v ex1x ex1.dat
Valid values Loc1 50004466 with geo mean: 36.7817
Valid values Loc2 49994581 with geo mean: 36.7826
File: ex1.dat with 1000001235 lines
  Command being timed: "ex1x ex1.dat"
  User time (seconds): 12.14
  System time (seconds): 0.67
  Percent of CPU this job got: 67%
  Elapsed (wall clock) time (h:mm:ss or m:ss): 0:18.98
  Average shared text size (kbytes): 0
  Average unshared data size (kbytes): 0
  Average stack size (kbytes): 0
  Average total size (kbytes): 0
  Maximum resident set size (kbytes): 792852
  Average resident set size (kbytes): 0
  Major (requiring I/O) page faults: 0
  Minor (reclaiming a frame) page faults: 10915
  Voluntary context switches: 7994
  Involuntary context switches: 64665
  Swaps: 0
  File system inputs: 4060288
  File system outputs: 0
  Socket messages sent: 0
  Socket messages received: 0
  Signals delivered: 0
  Page size (bytes): 4096
  Exit status: 0
```

[08.06.17, 16:18] bzfrehfe@opt59 ~/projects/lectures/appfs17/appfs/exercisel (master)
<92>
Exercise

Exercise 6: Extend your program for exercise 5 such that it measures (and prints) both its own wall-clock and user time.
Questions about exercise 5?