EECS 22L: Software Engineering Project in C Language

Lecture 5

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 5: Overview

- Course Administration
  - Wrapping up Project 1
  - Midterm exam
  - Course evaluation
- Software Development Tools
  - Linux commands and tools
  - GNU compiler tool chain
  - GNU profiling and debugging tools
  - Scripting languages, shells
  - Source code management, IDE

Course Administration

- Wrapping up Project 1
  1. **Software Release**:
     Final delivery due Monday, Feb. 11, 12pm (noon)
     - Binary program and documentation (**`Chess.tar.gz`**)
     - Source code and documentation (**`Chess_Source.tar.gz`**)
     - ➢ Refer to posted instructions for details on these packages!

  2. **Project Presentations**:
     Tuesday, Feb. 12, 9:30-11am (lecture slot, SE2 1306)
     - 10 minute team presentation (e.g. PowerPoint)
     - By one, few, or all team members
     - ➢ Title page (Team name, product, authors, …)
     - ➢ Main features of your software product
     - ➢ Main challenges encountered during software development
     - ➢ Main lessons learned

Course Administration

- Wrapping up Project 1 (cont'd)
  3. **Chess Tournament**:
     Tuesday, Feb. 12, 1-4:50pm (discussion + lab slots, EH 1141)
     - Every team plays their computer twice against all others
       - – 7 rounds, white vs. black and black vs. white
       - – In every round, 6 games are played in parallel
       - – Each team provides 2 program users and 1 observer
     - 10 minute maximum "thinking" time per player
       - – Each game lasts max. 20 minutes
     - Games may end in "white wins", "black wins", or "tie"
     - Team with most wins is the winner of the tournament

     - ➢ Detailed schedule provided by TAs!

Course Administration

- Wrapping up Project 1 (cont'd)
  4. **Midterm Exam**:
     Tuesday, Feb. 12, 1-4:50pm (discussion + lab slots, EH 1151)
     - 5 minute individual *oral exam* by instructor
       - Exams per team with members in alphabetical order
       - Scheduled during the "breaks" of the chess tournament

     ➢ Present your team's project, your contribution, and explain your source code (at the computer terminal)

     ➢ Oral Exam Questions:

     Q1: Briefly explain how your team's product works!

     ➢ What is the overall approach / algorithm?

     Q2: Which files in the software are you in charge of?

     Q3: Which part of your work was the most challenging?

     ➢ Why? Show your solution!

     Q4: Few ad-hoc questions on your code…

Course Administration

- Wrapping up Project 1 (cont'd)
  5. **Peer Evaluation**:
     Wednesday, Feb. 6, noon – Wednesday, Feb. 13, noon
     - Online EEE survey

     ➢ Mandatory, individual, confidential!

     ➢ Results will be seen only by the instructor and TAs!

     ➢ Questions:

     Q1: Name all your team members (including yourself) and estimate the *effort to the project* by each team member
       - Effort includes attendance, participation, coding, documentation, etc.
       - Scale of 1 ("poor") through 5 ("excellent")

     Q2: For project 2, choose up to 3 "buddies" you prefer to work with in the new team
       - New team size will be 7 to 9 students

Course Administration

- Wrapping up Project 1 (cont'd)
  6. **Midterm Course Evaluation**:
     Wednesday, Feb. 6, noon – Wednesday, Feb. 13, noon
     - Online via EEE Evaluation application
     - Voluntary, anonymous, confidential
     - ➢ Help to improve this class!
  7. **Start of Project 2**:
     Thursday, Feb. 14, 9:30-11am (lecture slot, SE2 1306)
     - ➢ New and larger teams!
     - ➢ New and more challenging topic!

Software Development Tools

- Linux Commands and Tools
  - Basic system commands [see EECS22 Lecture 1]
  - **`echo`** print a message
  - **`date`** print the current date and time
  - **`ls`** list the contents of the current directory
  - **`cat`** list the contents of files
  - **`more`** list the contents of files page by page
  - **`pwd`** print the path to the current working directory
  - **`mkdir`** create a new directory
  - **`cd`** change the current directory
  - **`cp`** copy a file
  - **`mv`** rename and/or move a file
  - **`rm`** remove (delete) a file
  - **`rmdir`** remove (delete) a directory
  - **`man`** view manual pages for system commands and tools

Software Development Tools

- Linux Commands and Tools
  - Text editors [see EECS22 Lecture 1]
  - **vi** standard Unix editor
  - **vim** vi-improved (supports syntax highlighting, and much more…)
    - Can compare two files and visualize the differences
      - **vi –d file1 file2**
  - **pico** easy-to-use text editor
  - **emacs** very powerful editor
  - **gedit** nice GUI editor in separate X-window

  - Manual page creation
  - **groff** simple text formatter
    - **groff -man -Tascii man/manl/name.l >man/catl/name.l**
  - Online how-to page: http://www.linuxhowtos.org/System/creatingman.htm

Software Development Tools

- Linux Commands and Tools
  - Advanced file system commands
  - **gtar** create (or inspect/extract) a “tar-ball” package
    - **gtar cvzf package.tar.gz files…**
    - **gtar tvzf package.tar.gz**
    - **gtar xvzf package.tar.gz**
  - **ln** create (symbolic or hard) links to files
    - **ln –s path_to_file link_name**
  - **chmod** set file access permissions
    - **ls –l filename**
    - **chmod u+rwx,g+rx-w,o-rwx filename**
    - **chmod 750 filename**
  - **groups** list group memberships of a user
  - **chgrp** change group for a file
    - **chgrp team7 filename**

Software Development Tools

- GNU Compiler Tool Chain
  - C/C++ Compiler GCC [see EECS22 Lecture 7]
    - **`gcc [options…] [files…] [options…]`**
  - Language Dialect
    - **`-ansi`** selects ANSI-C language semantics
  - Warnings
    - **`-Wall`** enables all warnings
  - Compilation phases
    - **`-E`** preprocessing only, result is preprocessed file (**`.i`**)
    - **`-S`** code generation only, result is assembly file (**`.s`**)
    - **`-c`** compilation only, result is an object file (**`.o`**)
    - (none) all compilation phases, result is executable
  - Output File
    - **`-o name`** selects output filename (default **`a.out`**)

Software Development Tools

- GNU Compiler Tool Chain
  - C/C++ Compiler GCC [see EECS22 Lecture 7]
    - **`gcc [options…] [files…] [options…]`**
  - Preprocessor definitions
    - **`-Dmacro`** command-line equivalent of **`#define macro`**
    - **`-Dm=def`** command-line equivalent of **`#define m def`**
    - **`-Umacro`** command-line equivalent of **`#undef macro`**
  - Support for debugging
    - **`-g`** generate symbol tables needed by debugger
    - **`–DDEBUG`** turn on conditional code for debugging
  - Optimization options
    - **`-O2`** optimize the generated code for speed (level 2)
    - **`-DNDEBUG`** turn off debugging support (i.e. assertions)

Software Development Tools

- GNU Compiler Tool Chain
  - C/C++ Compiler GCC [see EECS22 Lecture 7]
    - **`gcc [options…] [files…] [options…]`**
  - Input files for different stages
    - Module header file (file suffix **`.h`**)
    - Module program file (file suffix **`.c`**)
    - Module object file (file suffix **`.o`**)
    - Static or shared library file (file suffix **`.a`**, or **`.so`**)
  - Support for performance profiler **`gprof`**
    - **`-pg`** turns on profile instrumentation
  - Support for coverage testing tool **`gcov`**
    - **`-fprofile-arcs`** turns on flow graph support
    - **`-ftest-coverage`** turns on coverage support

Software Development Tools

- Execution Time Measurement
  - Linux kernel timing
    - **`/usr/bin/time command`**
- GNU Profiling Tools
  - GNU Profiler **`gprof`**
    - ➢ Find the critical performance bottleneck in a program
      - **`gcc –pg –g example.c …`**
      - **`./example`**
      - **`gprof example`**
  - GNU Coverage Testing Tool **`gcov`**
    - ➢ Determine code coverage when testing a program
      - **`gcc –fprofile-arcs -ftest-coverage –g example.c …`**
      - **`./example`**
      - **`gcov example`**

Software Development Tools

- Performance Profiling Example

```
doemer@ladera:1 > cd eecs22l/profiling/
doemer@ladera:2 > vi example.c
doemer@ladera:3 > gcc example.c -o example -ansi -Wall -g
doemer@ladera:4 > example
doemer@ladera:5 > /usr/bin/time example
4.39user 0.00system 0:04.40elapsed 99%CPU (0avgtext+0avgdata 0maxres.)k
0inputs+0outputs (0major+101minor)pagefaults 0swaps
doemer@ladera:6 > gcc example.c -o example -ansi -Wall -g -pg
doemer@ladera:7 > /usr/bin/time example
19.53user 0.00system 0:19.54elapsed 99%CPU (0avgtext+0avgdata
0maxresident)k
0inputs+0outputs (0major+126minor)pagefaults 0swaps
doemer@ladera:8 > ls
example*  example.c  gmon.out
doemer@ladera:9 >
```


Software Development Tools

- Performance Profiling Example

```
doemer@ladera:9 > gprof example
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 93.31      3.50     3.50        1     3.50     3.50  Fibo
  7.53      3.78     0.28       34     0.01     0.01  FmulN
  0.00      3.78     0.00       66     0.00     0.00  CountN

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.
[…]
```

Software Development Tools

- Performance Profiling Example

```
[…]
Call graph (explanation follows)

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    3.78                 main [1]
                3.50    0.00       1/1           Fibo [2]
                0.28    0.00      34/34          FmulN [3]
                0.00    0.00      66/66          CountN [4]
-----------------------------------------------
                                866988872             Fibo [2]
                3.50    0.00       1/1           main [1]
[2]     92.5    3.50    0.00       1+866988872 Fibo [2]
                                866988872             Fibo [2]
-----------------------------------------------
                0.28    0.00      34/34          main [1]
[3]      7.5    0.28    0.00      34          FmulN [3]
-----------------------------------------------
                0.00    0.00      66/66          main [1]
[4]      0.0    0.00    0.00      66          CountN [4]
-----------------------------------------------
This table describes the call tree of the program […]
doemer@ladera:10 >
```

Software Development Tools

- Coverage Testing Example

```
[…]
doemer@ladera:12 > rm gmon.out
doemer@ladera:13 > ls
example*  example.c
doemer@ladera:14 > gcc example.c -o example -ansi -Wall -g -fprofile-arcs
-ftest-coverage
doemer@ladera:15 > ls
example*  example.c  example.gcno
doemer@ladera:16 > /usr/bin/time example
4.81user 0.00system 0:04.82elapsed 99%CPU (0avgtext+0avgdata 0maxres.)k
0inputs+0outputs (0major+132minor)pagefaults 0swaps
doemer@ladera:17 > ls
example*  example.c  example.gcda  example.gcno
doemer@ladera:18 > gcov example
File 'example.c'
Lines executed:100.00% of 22
example.c:creating 'example.c.gcov'

doemer@ladera:19 >
```

Software Development Tools

- Coverage Testing Example

```
doemer@ladera:19 > ls
example*  example.c  example.c.gcov  example.gcda  example.gcno
doemer@ladera:20 > cat example.c.gcov
        -:    0:Source:example.c
        -:    0:Graph:example.gcno
        -:    0:Data:example.gcda
        -:    0:Runs:1
        -:    0:Programs:1
        -:    1:/* example.c: simple example to demonstrate profiling */
        -:    2:/* RD, 02/05/13. */
        -:    3:
        -:    4:void CountN(int n)
       66:    5:{
       66:    6:    int i, c = 0;
        -:    7:
    66066:    8:    for(i=0; i<n; i++)
        -:    9:    {
    66000:   10:        c = c + 1;
        -:   11:    }
       66:   12:}
[…]
doemer@ladera:21 >
```


Software Development Tools

- Debugging Tools
  - GNU Debugger [see EECS22 Lecture 10]
    - **gdb**
  - Data Display Debugger [see EECS22 Lecture 10 and 15]
    - **ddd**
  - Dynamic Memory Validator [see EECS22 Lecture 12]
    - **valgrind**
- Scripting Languages
  - Build scripts
    - **make**, **Makefile** [see EECS22 Lecture 8]
  - Cross-platform Make
    - **cmake**

Software Development Tools

- General Purpose Shell and Scripting Languages
  - Unix shell, and GNU bourne-again shell
    - **`sh`**
    - **`bash`**
  - Berkeley Unix C shell, and extension
    - **`csh`**
    - **`tcsh`**
- Remote Shells
  - Secure shell
    - **`ssh user@hostname.domain`**
    - **`scp user@hostname.domain:sourcefile targetfile`**
  - Insecure (!) remote shells
    - **`rsh`**
    - **`telnet`**

Software Development Tools

- Source Code Management
  - Concurrent Versions System [see EECS22L Lecture 3]
  - **`cvs checkout ...`**
  - Subversion
  - **`svn checkout ...`**
- Integrated Development Environment
  - **`eclipse`**
- Software Documentation Generator
  - **`doxygen`**