COMP 2355 Introduction to Systems Programming

Christian Grothoff
christian@grothoff.org

http://grothoff.org/christian/

README

http://grothoff.org/christian/teaching/2009/2355/

Academic dishonesty

- Course webpage says what is allowed.
- If in doubt, ask first!
- Cheating can be detected with automated tools.
- Any violation will be reported.

Definition: Plagiarism

Plagiarism is the presentation of another person's idea or product as your own. Plagiarism includes but is not limited to the following:

- Copying word-for-word all or in part of another's written work;
- using phrases, charts, figures, illustrations, graphics, codes, music, mathematical, scientific solutions without citing the source;
- except for common knowledge, paraphrasing ideas, conclusions, or research without citing the source;
- using all or part of a literary plot, poem, film, musical score, internet website or other artistic product without attributing the work to its creator.

Academic dishonesty

- You are encouraged to discuss the materials, homework, and projects together.
- However, all written assignments and programs must be done individually or in the assigned groups.
- Academic dishonesty includes, but is not limited to: plagiarism, cheating in exams, unauthorized collaboration and falsifying academic records.

Expectations

- Read the indicated chapters of the textbook – not every detail is covered in class, but it may still be part of the quizzes!
- Deliver tested, working versions of assignments on time using subversion.
- Review the material of the last class before the next class.
- Do well on in-class quizzes (no midterm, no final).
- Experiment. Participate in class.

Homeworks

- Programming assignments are posted on-line. You must submit those using subversion.
- Additionally, various smaller "homeworks" will be assigned in class (or done as practice during class).
- Those "homeworks" do not need to be submitted. However, questions relating to those homeworks will be asked during in-class quizzes.
- Experiment and explore. Doing just the bare minimum does not guarantee that you will have answers to all of the quiz questions!

Grades

$> \mathbf{90}$ A

$> \mathbf{75}$ B

$> \mathbf{60}$ C

$\geq \mathbf{45}$ D

$< \mathbf{45}$ F

Grading Policies

- Assignments: 60 pts
- Quizzes: 30 pts
- Final Exam / Project (Lecture 20): 10 pts
- No curve

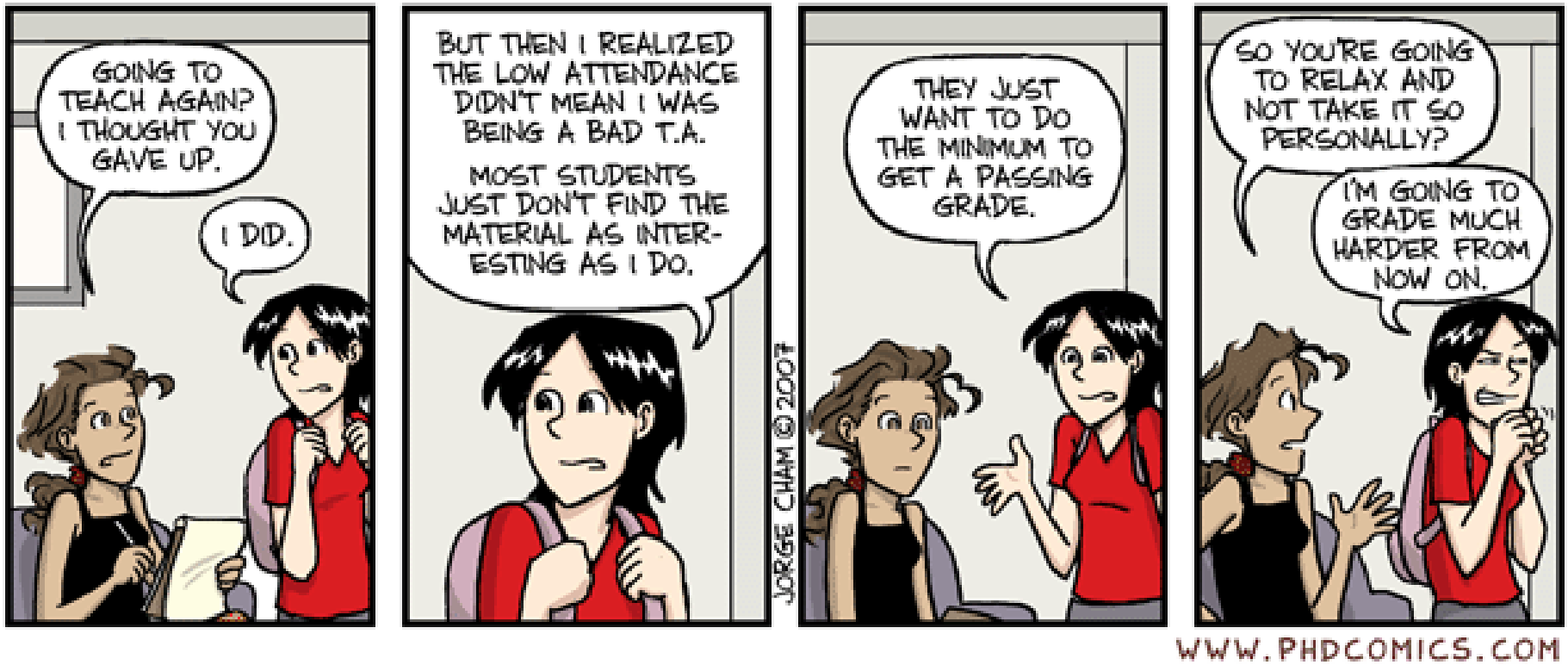
Workload

- This is supposed to be a hard course.
- You are expected to work on average about 12h/week in addition to the lecture for this class.
- If you do not understand the assigned material in the textbooks, ask well before the next lecture – it maybe on the quiz!

Quizzes

- Quizzes will be designed to test in-depth knowledge.
- Quizzes may ask questions related to any prior class, not just the previous class.
- Quizzes will cover textbook material and homework answers not discussed in class. Read and explore or fail!
- You can pass the class without the quizzes, if you do perfectly on the programming assignments

Clarification



Auditing Policy

- You can only audit this course if you submit all of the assignments
- Registered students have priority in office hours and with the TA
- The TA may or may not grade your projects

Hints from a Beta-Tester

- Have (install) your own GNU/Linux system
- You need to read the materials in detail; this will save you plenty of time when coding (less re-inventing the wheel!)
- This course is harder than programming languages, plan to spend more than 8h/week on it.

Content Overview

- The C programming language
- The C++ programming language
- System V APIs / GNU libc

Content (1/2)

- Datatypes in C
- The C preprocessor
- Linking and loading
- System V APIs
- Pointers
- C development and build systems

Content (2/2)

- Input-output in C
- Process management
- Higher order fun
- Terminal programming
- C++ namespaces, classes, templates, operators
- C++ STL
- Buffer overflows and memory corruption

COMP 3354

- Same lectures
- Additional assignment: Implement three CMS pipeline stages or suggest a good alternative project to the instructor
- Same grading percentages

Questions

?

Systems Theory

- Interdisiplinary field of science concerned with the study of complex systems
- A system is a set of interacting or interdependent entities.
- Managing complexity is the key problem in systems engineering.

What is Systems Programming?

Systems programming is the writing of system software.

Systems software is software supporting application software.

Examples of Systems Software

- Operating systems / Firmware
- Programming tools (compilers, linkers, loaders)
- (Essential) runtime libraries of programming languages

Why Study Systems Programming?

Systems programming is:

- essential to any kind of computing
- an important foundation for application development
- constantly evolving to accomodate changes in hardware (job security!)
- hacking with unparalleled freedom
- fun!

The Mindset of a Systems Programmer

- How does this really work (down to the hardware level)?
- How can I improve performance?
- I can do this – without trapeze and false bottoms!

Why Study C?

- C is the language of the systems programmer
- All major Operating Systems are written in C (BSD, Linux, OS X, Windows)
- C is a subset of C++ (so you cannot know C++ and not know C)
- C and C++ together are used for more than $\frac{1}{3}$ of all open-source projects at SourceForge[1]

[1] http://www.cs.berkeley.edu/ flab/languages.html

What is Special About C?

- C is essentially abstract assembly language of many historical processors (like the PDP series)

$\Rightarrow$ You can emulate features from high-level languages (such as Java, OCAML, PHP) in C!

- C has hardly any runtime system

$\Rightarrow$ Small footprint, easily ported to new architectures

Why Study C++?

C++ adds popular features from other languages to C.

\+ No need to emulate certain features manually

\+ Nice, standardized syntax for those features makes code easier to read

\+ Still all of the power of C "under the hood"

o Much larger language, harder to learn and use well

– C++ has a non-trivial runtime system

– C++ still evolves; C is much more stable as a language

Beyond "Introduction to C"

This course is also about good programming practice:

- Use of tools and principles behind those tools (subversion, compilers)
- Use of libraries and API design (GNU libc, STL)
- Understanding of UNIX design principles (System V, Security)
- Solving problems as a team

Collaboration

- Essential: problems are often too hard for just one person
- Different people contribute different skills
- Meeting in person is costly (time, travel, low productivity)

$\Rightarrow$ Internet-supported collaboration

Key Collaboration Tools

- Communication: E-mail, IRC, VoIP, ...
- Issue tracking: Forums, Bugtracking Systems, ...
- Knowledge integration: WWW pages, Forums, Wikis, ...
- Data management: Version Control Systems

Version Control Systems

Key Features of VCS include:

- Content Distribution
- Access Control Mechanisms
- Data Backup / Recovery / Rollback
- Branching and Merging

Content Distribution

- "Latest" version is in the VCS repository
- Any authorized user can obtain this version – possibly multiple times
- Before work starts, checkout latest version from repository
- Periodically during the session (and at the end), commit to repository

$\Rightarrow$ Easy way to keep data synchronized between multiple machines!

Normal Use

```
$ svn checkout https://svn/courses/comp2355/w2009/alice/
$ cd alice
$ mkdir P1
$ svn add P1
$ svn commit -m 'starting P1'
$ $EDITOR P1/mycode.c
$ svn add P1/mycode.c
$ svn commit -m 'first round'
$ $EDITOR P1/mycode.c
$ svn commit -m 'update'
```

Access Control

- Anonymous read-access: anyone can read the data
- Individual read access: specific users can read
- Individual write access: specific users can update
- Group access: simplify management by creating groups
- Partition repository: different rules for different directories

Authentication is usually done using username and password.

Example: Subversion Access Control

```
[/comp2355/w2009]
grothoff = rw
* =
[/comp2355/w2009/alice]
alice = rw
grothoff = rw
[/comp2355/w2009/bob]
bob = rw
grothoff = rw
[/comp2355/w2009/ateam]
alice = rw
bob = rw
```

Example: Subversion Access Control

The "passwd" file contains lines like this:

`grothoff:N2FHEWsLcoPto`

The SVN client transmits the password P. The server then computes $H(P + Salt)$ and compares with the hash code in "passwd".

Versioning

- Each `commit` operation creates a new revision
- VCS enables accessing all past revisions
- Subversion gives each revision a unique number (per repository)
- VCS attempts to minimize space overhead for storing revisions
- VCS enables concurrent editing and attempts to merge changes

Example: Concurrent Editing

1. Alice creates an initial text T_A and commits to the VCS (R1)
2. Bob retrieves T_A from the VCS and begins to edit
3. Carol retrieves T_A from the VCS and also edits it
4. Bob commits his updated text T_{AB} to VCS (R2)
5. Carol completes her edits (T_{AC}), but her commit fails: she edited R1, but the latest version is R2 (and she edited R1)

Example: Concurrent Editing

6. Carol retrieves Bob's changes $(T_{AB} - T_A)$ using VCS
7. The VCS automatically attempts to produce $T_{ABC} = T_{AC} + (T_{AB} - T_A)$.
8. If the VCS is not certain that it succeeded, it may require Carol to verify T_{ABC} manually.
9. Carol commits T_{ABC} as R3.
10. Alice requests the latest updates from the VCS, obtaining T_{ABC}.

Automatic Merging

- $T_{AB} - T_A$ is computed line-by-line
- Each change is stored with some context (lines before, lines after, offset in file, etc.)
- If changes apply to different lines and are at least a line apart in the document, automatic patching should succeed
- Otherwise, SVN produces a document with both versions

A Merge Conflict

```
Alices text.
<<<<<<<<<<
Bob inserted this text.
==========
Carol inserted this text.
>>>>>>>>>>
More text from Alice.
```

Edit the text to resolve the conflict, then use `svn resolved filename` to tell Subversion that all conflicts in the file have been addressed.

Branching

- Branches enable parallel development of closely related works
- Branches are created from a common starting point
- The starting point is often the current version, but does not have to be
- Each branch can make progress independently of the others
- VCS can help with merging branches
- For more information on branching, RTFM!

Questions

?

Homework hints

- $ svn add filename ; svn commit -m “logmessage”
- $ gcc -o binary sourcename.c ; ./binary

Homework summary

Before the next lecture:

- Generate password with `htpasswd` and register account.
- Read textbook chapters indicated on the webpage.
- Install software (or use department machines).
- Implement "Hello World" a few times.
- Test with provided script and submit!

Questions

?