CS 25200: Systems Programming

Lecture 1: Course Introduction, *NIX Commands

Prof. Turkstra

Lecture 01

- Objectives
- Course policies
- *NIX commands

Instructor

Dr. Jeffrey A. Turkstra

HAAS 128

jeff@purdue.edu

49-63088

Office Hours:
MW 10:00am – 11:00am
T 2:00pm – 4:00pm
OR
whenever door is open

...appointments also welcome

About me

- BSCmpE, MSECE, and PhD from Purdue University
  - Focused on operating systems and distributed systems
  - Instructor, ECE 2005-2008
  - Software Engineer with HUBzero/RCAC 2008-2017
  - Microfluidic Innovations, LLC and other startups
- Current academic activity
  - CS 252, CS 307, CS 240, CS 180, CS 250, CS 50011
  - Metachory
- Enjoys
  - Linux, skiing, piano/saxophone, astronomy, flying, HAM, etc
- More information at https://turkeyland.net/

Course information

- On the website
  - http://courses.cs.purdue.edu/cs25200:fall19:start
- Syllabus
- Lecture slides
- Schedule
  - Lectures may be different, roughly same schedule

Course topics

- Address space, program structure
  - Text, data, BSS, stack
- Use of an IDE and debugger
- Executable file formats
  - ELF, COFF, a.out, etc

- Development cycle, compiling, assembling, linking
- Static and shared libraries
- Loading a program
- Scripting languages, basic UNIX commands
- File creation, read, write, close, file modes
- I/O redirection, pipes

- Fork, wait, waitpid, signals
- Directories - creating, listing
- Write your own shell
- Programming with threads
- Race conditions, mutex locks
- Socket programming
- Iterative and concurrent servers

- Memory allocation – problems, leaks, premature frees, smashing, double frees
- Introduction to SQL
- Revision control systems
- Intro to software engineering
- Design patterns
- Execution profiling

Objectives

- Consolidate programming skills from previous core courses
- Understand how programs run in userland and interact with the OS
- Be able to write "large" programs with over 1,000 lines of code
- Be able to use tools – including IDEs, debuggers, profilers, and revision control – to create quality, maintainable code

- Learn to work in teams
- Be able to use and write shell scripts
- Learn how to write multi-process and multi-threaded programs

Grades

Projects	40%
Midterm	25%
Final	25%
Attendance	10%

Grading issues will be addressed as they occur. Not at the end of the semester.

Academic Integrity

- Do not cheat.
- Except for team projects, code should not be shared.
  - And then only among team members
- When in doubt, talk to a TA or instructor instead of a fellow student
- Protect your work
  - Having a publicly accessible git repo violates course policy

Questions and contact

- Piazza
  - https://piazza.com/purdue/fall2019/cs252
- Lab
- Office Hours
- Email
  - http://courses.cs.purdue.edu/cs25200:fall19:contacts

Labs

- Lab attendance is required and part of your attendance grade
- There is no lab the first week
- Projects will be explained in lab
- TA office hours will be posted on the course website

Quizzes

- Unannounced 5 to 10 minute quizzes
  - iClicker – be sure to register (if you haven’t already) by Wednesday (8/21)
- Score of 0 if absent

Projects

- There will be 6 projects/labs
- Individual efforts, except final project
- Reinforce lecture material as well as provide practical experience
  - Exploring low-level implementation details
- Deadlines on course website
  - Lab 1 posted soon
  - Due September 2, 11:58pm

Slides

- Slides based on Prof. Gustavo Rodgriguez-Rivera’s systems programming slides and Prof. Fred Mowle’s software engineering tools slides

*NIX Philosophy

- Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new "features".

- Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input.

- Design and build software, even operating systems, to be tried early, ideally within weeks. Don't hesitate to throw away the clumsy parts and rebuild them.

- Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you've finished using them.

In short

- “Do one thing and do it well”

man

- Interface to reference manuals

```
man [options] [section] file
```

- Examples…

```
man cp          # manual page for copy cmd
man -k printf # search pages for keyword
man 3 exec      # man page from section 3
```

Sections

- 1: Executable programs or shells
- 2: System calls
- 3: Library calls
- 4: Special files (usually in /dev)
- 5: File formats and coventions
- 6: Games
- 7: Miscellaneous
- 8: System administration commands
- 9: Kernel routines (non standard)

ls

- List directory contents

  ```
  ls [options] [file(s)]
  ```

  Examples…

  ```
  ls        # list files in pwd
  ls -al    # hidden (all) files, long listing
  ls -R     # recursive
  ```

mkdir

- Make directories

```
mkdir [options] directory
```

Examples…

```
mkdir bob
mkdir -p bob/is/deep # leading path
mkdir -m 700 bob     # set the mode
```

cp

- Copy files and directories

  ```
  cp [options] src dest
  ```

  Examples...

  ```
  cp bob joe
  cp -r dir1 dir2  # recursive
  cp -rp dir1 dir2 # preserve perms/times/etc
  cp -x dir1 dir2  # do not cross filesystems
  ```

mv

- Move (rename) files
  ```
  mv [options] src dest
  ```
  Examples…
  ```
  mv bob mike
  mv dir1 dir2
  mv -n bob exists  # do not overwrite
  mv -i bob exists  # interactive
  ```

rm

- Remove files or directories

  ```
  rm [options] file(s)
  ```

  Examples...

  ```
  rm a.txt b.txt
  rm -f a.txt  # force, no error message
  rm -r dir1   # recursive
  rm -rf dir1  # be careful

  rm -rf ~ /some/path   # be very careful
  ```

Questions?