Harvard Extension School
HARVARD DIVISION OF CONTINUING EDUCATION

CSCI E-28
Unix/Linux Systems Programming

Spring Term 2024

Course Information

CRN: 24040
Section Number: 1
Format: Flexible Attendance Web Conference
Credit Status: Undergraduate, Graduate
Credit Hours: 4

Course Description: As an introduction to the fundamental structure and services of the Unix and Linux operating systems, this course combines theory with programming at the system call level. Topics include files and directories, device control, terminal handling, processes and threads, signals, pipes, and sockets. Examples and exercises include directory management utilities, a shell, and a web server.

Prerequisites: Solid knowledge of C or C++ at the level of CSCI E-26 and a data structures course such as CSCI E-22; some experience using Unix helpful.

Instructor Information & Office Hours

Bruce Molay
Email: molay@fas.harvard.edu
Phone: 617 864 8832
Office Hours:
Saturday and Sunday each week at times to be based on student availability.
Additional Information:

Help is available during office hours, by on-line forum discussion, and by scheduling private Zoom meetings.

Section Meetings

Section meetings occur each week in Zoom to answer questions and to discuss ideas about the class content and the programming projects. Students are responsible for knowing any material presented during the section.   Attendance and participation are encouraged. If you are not available during the scheduled time, you are required to view the video of the meeting.

Course Goals / Learning Outcomes

Summary

Csci-e28 explains the structure of the Unix operating system and shows how to write system and network programs. It is appropriate for students who want to learn how to write system software for Unix/Linux or for students who want to learn about the structure of a multi-tasking, multi-user operating system.

The course covers the details of the file system, terminal and device input/output, multi-tasking, interprocess communication, video displays, and network programming.

Theory is presented in the context of how Unix implements the ideas. By the end of the course, students should be able to figure out how most Unix commands work and know enough about the system to draft their own version of most of them.

Preparation

You should be able to program in C or C++. You should be comfortable with pointers, structs, dynamic memory allocation, linked lists, and recursion.

You do not need to have programmed in C for Unix. If you know C++, you need to write in the C subset of C++.

Students are already expected to be comfortable with designing, coding, and debugging programs of modest complexity while employing good programming style, structured techniques, and employing appropriate data structures as appropriate.

Familiarity with Unix is helpful but not essential.

Mode of Attendance & Participation Policy

This class offers a live or on-demand option, which means you can choose to attend the class live over Zoom or watch the class recording afterward. You do not need to commit to the same mode of attendance for the whole semester.

If you are attending live over Zoom:

Class meetings take place over Zoom. Because they involve active participation, discussion, and dialogue, you are expected to attend all class meetings. Please arrive on time. You should attend Zoom meetings with a functional web-camera and microphone, prepared with materials needed, to engage thoughtfully, and with your camera on. You may turn off your camera for occasional interruptions or momentarily for privacy.

You will also need the most up-to-date Zoom client installed on your computer to join class. Please participate from a safe and appropriate environment with appropriate clothing for class. Participating while traveling or in a car is not permitted. In addition, please do not join class via mobile phone or web browser.

If you are participating on demand:

You are expected to watch the class recording and complete any assignments before the next live class meets.

Please be sure to review important information on **Student Policies and Conduct**.

Grading & Grade Definitions

Grading

1% Skills Check

Homework 0 skills check Due Jan 31

4% Class Participation

Students are strongly encouraged to attend class during the live presentation and to participate by asking and answering questions. People who cannot attend class may participate on the Ed discussion site, the weekly discussion sections, and weekly office hours.

60% Assignments / Problem Sets - DUE:

Assignment more03 Due Jan 28

Homework 1 Due (Feb 11)

Homework 2 Due (Feb 25)

Homework 3 pstty Due (Mar 10)

Homework 4 pong Due (Mar 24)

Homework 5 Due (Apr 14): smallsh

Homework 6 Due (Sun Apr 28): network project

35% Final Exam DUE: May 8, 2024

The final exam is an online proctored exam. The final exam is open notes and open book, but you may not use any electronic devices for reference or assistance during the exam.

Your grade for the course should reflect what you know and can do as the course ends. I think of your grade as a very brief letter of recommendation to your next instructor or a possible employer. Therefore, I may adjust these fractions if your final exam shows very significant improvement or very significant decline relative to your assignments.

If the final exam shows significant improvement or if your work on the final exam is significantly weaker than your work on programming assignments, I shall put more weight on the final exam so the course grade reflects more accurately where you are at the end.

Four Late Days and One Catastrophe

Homework is due by midnight on Sunday evenings. There is a 10 point penalty for each day late you turn in an assignment. Students are allowed four free late days and one catastrophe. We drop the lowest good faith submission, even if it is very late. This allows for one catastrophe without penalty. Not turning in an assignment is not dropped.

Programming Project Grading

The assignments are graded on a 100-point scale. 70% is for correct operation of the program. 30% is for design, readability, efficiency, and documentation.

Additional Details about Projects

For additional details about the projects and programming requirements, visit https://cscie28.dce.harvard.edu/~dce-lib215/

Grade Definitions

Students registered for undergraduate or graduate credit who complete the requirements of a course may earn one of the following grades:

A and A– Earned by work whose superior quality indicates a full mastery of the subject —and in the case of A, work of extraordinary distinction. There is no grade of A+.

B+, B, and B– Earned by work that indicates a strong comprehension of the course material, a good command of the skills needed to work with the course materials, and the student's full engagement with the course requirements and activities.

C+, C, and C– Earned by work that indicates an adequate and satisfactory comprehension of the course material and the skills needed to work with the course materials, and that indicates that the student has met the basic requirements for completing assigned work and participating in class activities.

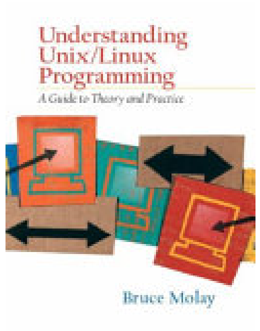
D+, D, and D– Earned by work that is unsatisfactory but that indicates some minimal command of the course materials and some minimal participation in class activities that is worthy of course credit.

E Earned by work that is unsatisfactory and unworthy of course credit. This grade may also be assigned to students who do not submit required work in courses from which they have not officially withdrawn by the withdrawal deadline. Zero or E grades are assigned to students for missing work. These grades are included in the calculation of the final grade.

Graduate Credit Requirements

Graduate credit students are required to submit additional software design/planning documents as part of their programming projects.

Course Materials


Understanding Unix/Linux Programming
A Guide to Theory and Practice
Bruce Molay

Understanding Unix/Linux Programming
Authors: Bruce Molay
Understanding Unix/Linux Programming is the required main text. This book follows the course closely.
Publisher: Pearson
Publication Date: 2003-01-01

Academic Integrity Policy

You are responsible for understanding Harvard Extension School policies on Academic Integrity and how to use sources responsibly. Violations of academic integrity are taken very seriously. Visit Using Sources Effectively and Responsibly and the Harvard Guide to Using Sources to review important information on academic citation rules.

AI Technologies. The Extension School's Academic Integrity Policy. prohibits students from representing work as their own that they did not write, code, or create.

Course Goals: CSCI-E28 is a programming course using C and Unix. The goals of the course are to help you learn about systems programming for Unix/Linux systems, and **also** to improve your programming and design skills. In the same way that using Google Translate to do assignments for a course in French language and culture prevents students from actually learning French language and culture, using Chat-GPT or other generative AI system to understand/call system services, design algorithms. use data structures, and solve problems prevents you from learning how to understand/call system services, design algorithms, use data structures, and solve problems.

In order **to achieve these goals**, we expect students to practice syntax, algorithm design, and problem solving. We expect that all work students submit for this course will be their own. We specifically forbid the use of ChatGPT or any other generative artificial intelligence (AI) tools at all stages of the work process, including preliminary ones. Violations of this policy will be considered academic misconduct. We draw your attention to the fact that different classes at Harvard could implement different AI policies, and it is the student's responsibility to conform to expectations for each course.

Writing Code. While it may be common practice in non-academic settings to adapt code examples found online or in texts, this is not the case in academia. In particular, you should never copy code produced as coursework by other students, whether in the current term or a previous term; nor may you provide work for other students to use. Copying code from another student or any other source is a form of academic dishonesty, as is deriving a program substantially from the work of another.

Writing code is similar to academic writing in that when you use or adapt code developed by someone else as part of your assigned coursework, you must cite your source. Paraphrasing without proper citation is just as dishonest with programming as it is with prose. A program can be considered plagiarized even though no single line is identical to any line of the source.

Accessibility Services Policy

The Division of Continuing Education (DCE) is committed to providing an accessible academic community. The Accessibility Services Office (ASO) is responsible for providing accommodations to students with disabilities. Students must request accommodations or adjustments through the ASO. Instructors cannot grant accommodation requests without prior ASO approval. It is imperative to be in touch with the ASO as soon as possible to avoid delays in the provision of accommodation.

DCE takes student privacy seriously. Any medical documentation should be provided directly to the ASO if a substantial accommodation is required. If you miss class due to a short-term illness, notify your instructor and/or TA but do not include a doctor's note.

Course staff will not request, accept, or review doctor's notes or other medical documentation. For more information, email accessibility@extension.harvard.edu.

Publishing or Distributing Course Materials Policy

Students may not post, publish, sell, or otherwise publicly distribute course materials without the written permission of the course instructor. Such materials include, but are not limited to, the following: lecture notes, lecture slides, video, or audio recordings, assignments, problem sets, examinations, other students' work, and answer keys. Students who sell, post, publish, or distribute course materials without written permission, whether for the purposes of soliciting answers or otherwise, may be subject to disciplinary action, up to and including requirement to withdraw. Further, students may not make video or audio recordings of class sessions for their own use without written permission of the instructor.

Class Meeting Schedule

Section 1: Files

January 24: Class 1 Topic: Introduction to Systems Programming

January 31: Class 2 Topic: Files, System Files, File Formats, Buffered I/O

February 7: Class 3 Topic: Directories, File Info, Bits

February 14: Class 4 Topic: File Systems

Section 2: Device I/O

February 14: Class 5 Topic: Files, Devices, Drivers

Section 3: Multi-Tasking

February 28: Class 6 Topic: Writing a video game: curses, timers, polling

March 6: Class 7 Topic: Processes: Writing a Shell: fork, exec, wait

March 13: NO CLASS (Spring Break)

March 20: Class 8 Topic: A Programmable Shell, The Environment

Section 4: Inter-Process Communication

March 27: Class 9 Topic: I/O Redirection and Pipes

April 3: Class 10 Topic: Network Programming: Sockets, Servers, Clients

Section 5: Network Programming

April 10: Class 11 Topic: Network Programming: A Web Server

April 17: Class 12 Topic: Network Programming: License Server

April 24: Class 13 Topic: Concurrent Programming

May 1: Class 14 Review

May 8: Final Exam

Final Exam

The final exam will be given May 8, 2024. The exam is proctored and written/typed using Canvas and Proctorio. You may bring paper reference material (notes and books) but you may not use any electronic devices (computer, tablet, phone, etc.) during the exam.

Self-Assessment

Am I Prepared for the Course?

The purpose of this assignment is to review basic Unix and C skills and help you figure out if you are prepared for this course. Work through the following exercises. Feel free to refer to any Unix/C books and/or on-line documentation you like but design and code the programming exercises yourself.

You should find the C exercises pretty easy. If you don't find them pretty easy, think carefully before enrolling. If you can solve them, that does not ensure you are prepared for the course. The projects for the course are much larger and more complicated than these short programs.

You should find the Unix exercises easy or you should be able to locate the information in the manuals. If you don't, find a good Unix book and start exploring.

You may email solutions to molay@fas.harvard.edu for advice about your level of preparation and/or to ask any questions about the course..

The Exercises

1. Explain the purpose of the following Unix commands: ls, cat, rm, cp, mv, mkdir, cc.

2. Using your favorite editor, create a small text file.

Use cat to make five copies of that file.

Use wc to count the number of characters and words in the original file and in the one you made from it. Explain the result.

Create a subdirectory and move the two files into it.

3. Create a file containing a directory listing of both your home directory and the directory /bin.

4. Devise a single command line that displays the number of users currently logged onto your system.

5. Write, compile, and execute a C program that prints a welcoming message of your choice.

6. Write, compile, and execute a C program that prints its arguments.

7. Using getchar() write a program that counts the number of words, lines, and characters in its input.

8. Create a file containing a C function that prints the message "hello, world". Create a separate file containing the main program which calls this function. Compile and link the resulting program, calling it hw.

9. Look up the entries for the following topics in your system's manual; the cat command, the printf function, and the write system call.

10. You Must Try This One There are two parts to the problem. If you are not able to do the first part of this problem, you are not prepared to take this class. If you find the second part extremely tricky, you will find the assignments for the course difficult and potentially more time consuming than you expect

part 1 Write a program that prints a range of lines from a text file. The program should take command line arguments of the form:

```
lrange 10 20 filename

or

lrange 10 20
```

will print lines 10 through 20 of the named file. If there is no named file the program prints lines read from standard input. If there are not enough lines in the file, the program should print what it can.

Your program should work for input with any number of lines. Your program should work for lines of any length.

part 2 Write a program called last10 that prints the last ten lines of a text file. The program can be used from the command line with:

```
last10 filename
```

If there is no filename, last10 processes standard input.

Your program should work for input with any number of lines.

*11. Structs and Pointers*You must try this one, too. Write a function that computes some basic statistics for a list of numbers and stores those results in parts of a struct. In particular, given this definition:

```
struct numlist
{   float *list; // points to list of numbers
    int len;     // number of items in list
    float min,   // the minimal value in list
          max,   // the maximal value in list
          avg;   //
};
```

-

write a function compute_stats(struct numlist *listptr) that takes as an argument a pointer to a struct numlist with list and len already initialized and computes and fills in the other three members. Write a program that uses this function to process user input and print results.

12.

- *Dynamic Memory Management*This one is an important skill. Write a C program that reads in an arbitrary number of lines then prints those lines in reverse order. The lengths of the lines may be limited to a fixed maximum. but the number of lines is limited only by system memory available.
  For a slightly greater challenge, support a -b command line option that causes each line to be printed out backwards.

  The program, like most Unix tools, reads from a file if a filename is given or from standard input if no filename is given on the command line.

  Running this program on its own output should produce the original input. For example

  ./reverse /etc/passwd **|** ./reverse > rev2diff rev2 /etc/passwd should show no difference

Participation Counts for 4% of the Course Grade

Please be sure to review the following important information about Student Policies and Conduct.

Students are strongly encouraged to attend class during the live presentation and to participate by asking and answering questions. People who cannot attend class may participate on the Ed discussion site, the weekly discussion sections, and weekly office hours.

Class participation counts for 4% of the course grade.