CS 306/491-3 – Linux Programming – Fall 2015
MWF 3:00–3:50 p.m., Parkinson 202

See the course web page for more information and resources: http://www.cs.siu.edu/~cs306

Professor: Dr. Norman Carver, Faner 3121, phone 453-6048, email carver@cs.siu.edu.
Office hours: MWF 12:30–2:00pm and 4:00–4:30pm (or by appointment).

TA: Mounika Atluri, email: mounikaatluri@siu.edu
Office hours: TTh 1:00–3:00pm (meeting in TA Room, Faner 2131)

Required Text: (choose either of the following)
(1) GNU/Linux Application Programming (2nd ed) by M. Tim Jones.
(2) Beginning Linux Programming (4th ed) by Matthew and Stones.

Workload: labs and homework (required for exam admittance!), quizzes (10%), three exams (20% each), final exam (comprehensive, but optional 30%).

Dates: vacations: 9/7, 10/12, 11/11, 11/25–27; last drop date: 11/1 (online);
final exam: Fri., 12/18, 2:45–4:45pm.

Goals of the Course
• Become comfortable using basic Linux/UNIX commands and utilities.
• Introduce the C language and get experience programming in C.
• Learn how to develop software on and for Linux/UNIX systems.
• Become familiar with important Linux/UNIX library functions and system calls.
• Introduce asynchronous and concurrent (multi-process/multithreaded) programs.
• Introduce network programming.
• Lay a foundation for an operating systems course.

Basic Overview of Course Topics*
1. Introduction to operating systems and Linux/UNIX
2. Effective Bash shell (CLI) usage
3. Introduction to the C language
4. Linux/UNIX programming and development tools
5. Systems programming
6. Concurrent programming
7. Network programming

*See the course web page for a more detailed outline of topics to be covered, along with the relevant textbook sections, slides, and handouts.
Key Points about Course Policy:

- This course is being offered in what is termed a flipped classroom format. Students are to watch online video lectures on their own, with class time then spent on the homeworks and labs.

- A schedule of the dates when particular lectures need to have been watched will be on the course website. You must watch the videos by the specified date or you will not fully understand what is being done in class.

- To encourage students to watch the lectures on time, there will be weekly quizzes on the scheduled material, worth 10% of the final grade.

- 90% of your final grade will be based on the exams, which will include both short answer questions and questions that require you to write code.

- Completing the lab (programming) assignments is absolutely critical to learn what you must from this course, as well as for doing well on the exams.

- Turning in sufficient working labs will be required to be allowed to sit for exams!

- It is essential that you start on the labs early enough so that you can complete the lab before the due date; due dates will be firm deadlines (late submissions will not be accepted).

- Failing to turn in a working lab means a zero on the associated exam, with severe consequences for your final grade in the course.

- Note that lab scores will not directly enter into your final grade, since they are intended to be a learning experience, and because students enter this class with a range of programming background.

- Labs can be done in teams of up to three students if desired.

- Turning in any code not written by you or your teammates will be considered cheating; you will get a zero for the assignment and thus on the relevant exam!

- All development is to be done on a Linux platform; any Linux distribution should work fine (Windows will not).

- Linux can be run on top of Windows using Virtual Machine software; VirtualBox images of select distros will be made available via the course website.
Flipped Classroom:

- This course is being offered in what is termed a “flipped classroom” format. The basic idea behind this approach is that lectures on course material are provided by having students watch online video lectures on their own, with class time then spent primarily on improving student skills needed for the homeworks and labs.
- The most significant problems students have been having in this course involve weak problem solving and programming skills. Using a flipped classroom approach addresses this by vastly increasing the amount of interaction on these issues.
- This approach does require that students keep up with the video lectures (and textbook chapters). Students who have failed to study the appropriate material will generally have a difficult time understanding in-class discussions, since these will involve using functions and techniques they are unfamiliar with.
- A schedule of the dates by which particular lectures need to have been watched is available on the course website.
- The schedule takes into account that for a 3 credit hour course, a student is supposed to attend 3 hours of lectures and do a minimum of 6 hours of outside of class study each week.
- To encourage students to keep up with the online lectures, there will be weekly quizzes! The quizzes represent a significant (10%) portion of your overall score; failing to keep up with lectures could potentially cause a drop of one letter grade.
- The lectures (and associated slides) are available through the course website. Most lectures have been broken down into approximately 20 minute segments, making it easier to fit watching them into one’s schedule, and easier to review particular elements. The slides for each lecture topic can also be used for review or as reference material.
- Note also that while we will be working on the lab/homework topics in class, students will still have to complete the assignments outside of class.

Prerequisites, Attendance, and Textbooks:

- The CS Dept prerequisite will be enforced: CS 220 w/C or better, or equivalent. ECE/CEGR students are to have taken ECE 222 and ECE 321.
- You are not required to be familiar with the C language or with the Linux OS.
- You are expected to be familiar with programming in at least one C-family language (Java, C, C++), through the level of a standard CS II course (i.e., data structures).
- Attendance at the scheduled class sessions is a very important part of this course for almost all students! While students are to watch the lectures on their own, the class sessions will focus on improving students abilities to program in C on Linux systems. For many students, successful completion of the lab assignments will be possible only with the assistance provided via the class sessions.
- Having access to one of the textbooks is considered a requirement of this course, and you are expected to read the assigned sections in one of the course textbooks. You may be tested on assigned material in the texts, even if it has not been presented in lectures.
Lab/Homework Assignments Permit Access to Exams:

- In this course, the scores you obtain on the labs/homeworks are not a direct component of your final course score.
- Instead, submission of suitable lab/homework assignments will be required for you to be allowed to sit for each exam!
- If you fail to submit a suitable version of an assignment, you will not be allowed to sit for the associated exam. This means that you will receive an automatic zero for that exam (making it very difficult to achieve a C grade for the class).
- Lab (programming) and homework assignments play a crucial role in students learning the concepts they are supposed to from this course. Before this lab scoring policy was instituted, significant fractions of students were failing to complete the programming assignments. Not only were they then failing the coding problems on the exams, they were failing to learn what they were supposed to from the course.
- Lab submission rates (and course pass rates) have gone up significantly since this policy was put into place! So whether you like it or not, the approach has served its purpose well, and will be strictly enforced!!
- Though the precise scores you receive on lab/homework assignments will not directly affect your final grade, assignments will still be graded so that you can see what you know and what you do not.
- **You must score at least 40% on an assignment for your submission to count toward exam admittance!** Achieving such a score will require that your program compile and run, performing at least some of the assignment’s functionality.
- Some students complain that this 40% requirement eliminates the incentive for trying to do well on the labs, but this is not true in practice. First, it is much more likely that a student who is able to score 100 on a lab will fully understand the concepts being taught on the lab, resulting in a higher score on the exams. Second, students virtually always overestimate the quality of their submissions. This makes it a risky gamble to shoot for much less than a 100, since you do not know precisely how we will score your submission.
- Furthermore, there are several excellent pedagogical reasons for using this approach. First, the labs are intended as learning experiences (while the exams are intended to evaluate your knowledge). It is desirable to allow a student to get only a 40 on a lab because he may not understand some key concepts, but then put the effort into learning what he did not understand prior to the exam, and as a result suffer no ill effects. When lab scores are a direct component of the final grade, students can get penalized for concepts they don’t even realize they do not completely understand.
- Second, students enter this course with varying programming backgrounds, especially varying C language skills. Not too surprisingly, those with more C programming background tend to do better on the labs. By allowing the labs to be for learning and the exams for evaluation, the playing field is leveled for those that are willing to work to learn the material in this course.
• Third, the lab scoring policy permits lab assignments to be more challenging. Assignments have more pedagogical value when students are challenged, even if that means it is difficult to achieve a perfect score. When lab scores affect the course grade, assignments must be easy enough for most students to get “good” scores. This often leads to assigning short programs that involve only “spoon fed” concepts. Unfortunately, that is a very poor way to learn how to program and problem solve for the real world. (More on this below.)

• Finally, another benefit of the lab scoring approach is that it allows students to learn collaboratively, while minimizing the effects of “cheating.” (More on this below.)

Grading:

• Your course grade will be based primarily (90%) on exam performance!

• Each exam will consist of both short answer and coding (programming) problems. It is an important goal of this course that you learn to program in C, using system calls, so you will be required to demonstrate this skill by writing code on the exams. The coding problems will constitute about 50% of the total points on each exam.

• There will be three exams during the semester. The Final Exam will be comprehensive, but optional for students satisfied with their grades from the three exams.

• See the course webpage for a list of the assignments that must be submitted to be allowed to sit for each of the exams.

• Lab/homework assignments will not be accepted for scoring after the final due date/time has been set, unless a student can document a significant reason why the assignment could not be submitted on time. The only exception to this policy is that students will be granted a 24 hour “grace period” to submit an assignment (e.g., for “illness”).

• Makeup exams will be offered only for significant and documented conditions such as illness or university-related travel, at the discretion of the instructor.

• Final course letter grades will be “curved”: e.g., rather than A’s being limited to score averages over 90%, an average of 87% might result in an A. However, the only way to guarantee, say, an A, is to achieve an average exam score of better than 90%. Students must generally end up with an average of at least 60% to receive a grade of C or better in this course (which is what is required by both the BS and BA CS degrees).

Lab (Programming) Assignments:

• While the lab assignment scores will not be a direct component of your grade, the labs play a critical role in acquiring the skills that are to be learned from the course. It is simply impossible to learn to program well without spending significant time programming! Virtually no students do C or better work on the exams unless they have invested enough time on the programming assignments.

• Because of the role that assignments play in allowing you to sit for exams, it is critical to your success in this course that you begin working on each lab assignment as soon as possible after it is assigned, so you complete it by the deadline.
• Programs that do not compile or do not run through at least some required functionality, will not be graded, and will receive an automatic zero score! We are not going to spend vast amounts of time trying to figure out how much of an assignment you completed by examining source code, when you were not willing to spend enough time to complete the assignment.

• One of the most difficult aspects of software development is estimating how long it will take to complete a program (including debugging). Even professionals routinely underestimate the time that will be required. Do not simply guess that a lab will take you around four hours, then wait to start until 8pm the night before the lab is due. Not only can such a decision lead to you failing this class, the instructors have read enough emails sent by students at 4am the day a lab is due to know just how distressing it is to be discovering the “joys” of pointers and the like under deadline conditions.

• Getting your program to compile should not be viewed as a significant accomplishment. Compilation errors indicate that you do not even understand the most basic aspect of the programming language: its syntax. Being able to compile your code gives not the slightest indication of whether the program’s logic is correct. Almost the only reason an experienced program has compilation errors is because of typos!

• When working in an unfamiliar language or on unfamiliar problems, it is important to try to minimize the number of errors in your code at any one time. The best way to do this is to work incrementally (i.e., in stages): start with some basic functionality, get that working, then add more of the required functionality, get that working, and so forth. Resist as much as possible any urge you have to type your entire program in and then start trying to debug it. Such an approach makes failure much more likely!

• Lab (programming) assignments will be provided in relatively detailed program specification documents. You should treat these document as if they are being provided by your “boss” and your job depends on how well your programs adhere to the specs. Programming provides many avenues for creativity, but not when it comes to a program spec; you are free to meet the spec any way you want, but you must provide a program that is consistent with the spec. Far too many students lose points on their assignments because they fail to fully read the specification or because they fail to take it seriously. For example, if you are told to use a certain prompt or format output in a certain way, then if you choose to use a different prompt or format the output differently, your program is incorrect and points will be lost.

• Some students complain that the lab assignments are “too challenging” or too time consuming, suggesting that we should instead do a larger number of simpler assignments. Simpler assignments lead to substantially less learning, however. The companies that hire CS graduates routinely tell us that our students need to be required to implement substantive programs, multiple pages in length, since this is the kind of coding they will face in the “real world.” While you might prefer having to complete only one to two page programs for this course, those programs would not prepare you to be successful once you graduate. One of the advantages of the lab/homework grading policy in this course is that you are not required to always turn in perfect programs in order to do well in the course. It is perfectly OK to struggle with an assignment, coming to understand it fully only before the exam.
Platforms and Facilities:

- Programming for the course will be done in C (not C++).
- The official OS platform for the course is Linux and the official C compiler is GCC.
- Students are neither required nor expected to have a personal Linux machine available.
- Students will be provided account so they can use the CS Department’s Linux workstations, which are located in the CS Dept. Linux Lab (Faner 2102).
- Be aware that the CS Dept. Linux Lab is not open evening and weekends! If you need to do your work on the CS Dept. workstations, be sure to plan your time accordingly.
- Two CS Dept. Linux workstations are remotely accessible via SSH: pc00.cs.siu.edu & pc01.cs.siu.edu. While this may allow you to work on assignments even when the Linux Lab is closed, be aware that you are responsible for figuring out how to remotely access these machines and transfer files, and you will not be excused from getting assignments in on time if you encounter access problems (unless the machines cannot be accessed due to SIUC or CS Dept. problems).
- For students who do want to run Linux on a personal machine, we make virtual machine images for select Linux distributions available for you to run with the free VirtualBox virtualization software. Instructions for downloading and setting up these images is available on the course website.
- Note that this course does not cover Linux installation/configuration (see CS 406). Support cannot be provided for for students attempting anything but the VirtualBox image setups.
- While the CS Dept workstations currently run Ubuntu Linux, virtually all Linux distributions should be compatible and will have GCC and other development tools available (though these tools may not be installed by default). Other UNIX-like OS’s (e.g., Mac OS X, a BSD, or Solaris) should also be usable, but are not officially supported. If you choose to use one of them instead of Linux, it is up to you to make certain your programs compile and run properly on the CS Dept. Linux workstations before you submit them.
- Windows OS’s are definitely not compatible and should not be used for development in this course. Most of the course will involve Linux/UNIX system programming, so it will be impossible for you to compile or run your programs under any version of Windows OS.
- You are strongly encouraged to do all your work for the course on Linux machines (either the CS Dept workstations or a personal Linux machine). This will help you become familiar with the Linux commands that you will have to know for the exams, and will avoid compilation and other problems. Students that regularly use Linux systems for their coursework always perform better on the exams than those who choose to work via Windows machines.
- You will be instructed how to electronically submit your labs/programs for grading. Failure to follow the instructions may result in a score of zero for labs.
Cheating versus Acceptable Collaboration:

- *Cheating* on programming assignments is a significant problem throughout CS courses. The CS Dept has a webpage on *Academic Dishonesty* on its website, and this defines cheating in some detail. You should familiarize yourself with it.

- *If you cheat on an assignment in this class, you will receive a zero for the assignment.* This will in turn cause you to fail to meet the requirements for admission to one exam, which will severely impact your course grade. *A second violation will result in an immediate ‘F’ for the course.*

- While collaborating on assignments with classmates is considered cheating in many CS classes, the grading structure for this course allows for limited collaboration on the programming assignments.

- You will be allowed to work on lab (programming) assignments for CS 306 in teams of no more than three students. If you choose to work in a team, every team member must confirm the team by:
  1. sending the instructor an email listing his fellow teammates prior to submission;
  2. including a *comment* at the top of submitted source file(s) listing all teammates.

- While some collaboration on lab assignments is being allowed, this does not necessarily mean that collaboration is the best way for you to work. If you are capable of completing the assignments by yourself, you will probably learn more and be better prepared for the exams than if you work in a team. On the other hand, developing code in conjunction with other students has been shown to be helpful for some students (if done properly). The danger is that some of the team members will do most of the work, resulting in the weaker member(s) failing to learn what they must. If you work collaboratively, make certain that you understand every single line of the code that your team submits, or you are likely to fare poorly on the exams.

- Every line of code that you submit must have been written by you or your teammates (or have been provided by the instructor with the understanding that you could copy his code).

- If this is not the case, then you have cheated on the assignment!

- Incorporating code that somebody else wrote into your own solution is not allowed, whether that code came from a classmate (other than a teammate), a friend, the instructor (without permission), the TA, or the Internet. Starting with somebody else’s code is still cheating even if you slightly modify that code—such as by changing variable names, swapping the orders of a few statements, etc.

- If you don’t understand why submitting somebody else’s modified code is cheating, consider that this would be comparable to taking a book somebody else had written, changing the names of the characters, and then claiming the “new” book as your own. Very few people would consider that to be true.

- In fact, because computer software is inherently *copyrighted* in the US and most of the rest of the world, copying somebody else’s code without their permission is not just cheating, it is a *violation of law.*
- This is even the case for most “free and open source software,” since virtually all such software requires that anyone using the code in any way must retain existing authorship and copyright notices in any derived code (which would obviously alert the instructor that you had cheated).

- Studies of cheating in CS classes have found that students drastically underestimate the ease with which instructors can identify copied code. No two independently developed programs of any length should look terribly similar since there will be a vast number of alternative ways to meet the program specifications. Students also seem to forget that the instructor/TAs can perform the very same Internet searches that they can, and so can find the same publicly available code.

- The proper way to use the Internet to assist with programming is to use it to get ideas about how to implement certain functionality that you do not understand. You are free to use these ideas in your implementation. If you find yourself cutting and pasting somebody else’s code into your program, you are almost certainly cheating (even if you then modify it slightly afterwards).

**Emergency Procedures:**
Southern Illinois University Carbondale is committed to providing a safe and healthy environment for study and work. Because some health and safety circumstances are beyond our control, we ask that you become familiar with the SIUC Emergency Response Plan and Building Emergency Response Team (BERT) program. Emergency response information is available on posters in buildings on campus, available on BERT’s website at www.bert.siu.edu, Department of Safety’s website www.dps.siu.edu (disaster drop down) and in Emergency Response Guideline pamphlet. Know how to respond to each type of emergency.

Instructors will provide guidance and direction to students in the classroom in the event of an emergency affecting your location. It is important that you follow these instructions and stay with your instructor during an evacuation or sheltering emergency. The Building Emergency Response Team will provide assistance to your instructor in evacuating the building or sheltering within the facility.