

Grading

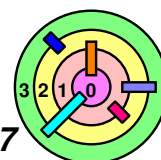
➡ Quizzes 5% 7/8 quizzes (we will drop the lowest two grades)

➡ Projects 45% 2 warm-ups (individual), 3 group projects

➡ Midterm 25%

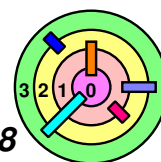
➡ Final 25%

- it's your responsibility to make sure that you don't have another final exam during this time
 - if you are enrolled in a class with same lecture time but on Friday, check with the other professor to make sure their final exam time doesn't conflict with ours
 - ◆ if there is a conflict, you must contact me to make a request to be "*virtually switched*" to another section and take both midterm *and* final exams in another section



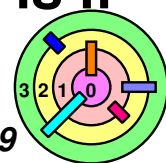
Grading

- ➡ Extra credit
 - 1) turn in *assignments* more than 48 hours before deadline
 - 2) giving *good/useful/timely* answers in the class Google Group for *kernel assignments*
- ➡ extra credit are just "extra"
 - you keep what's over 100%
- ➡ No *individual* extra credit (due to my fairness policy)
 - ➡ try your best from the beginning!



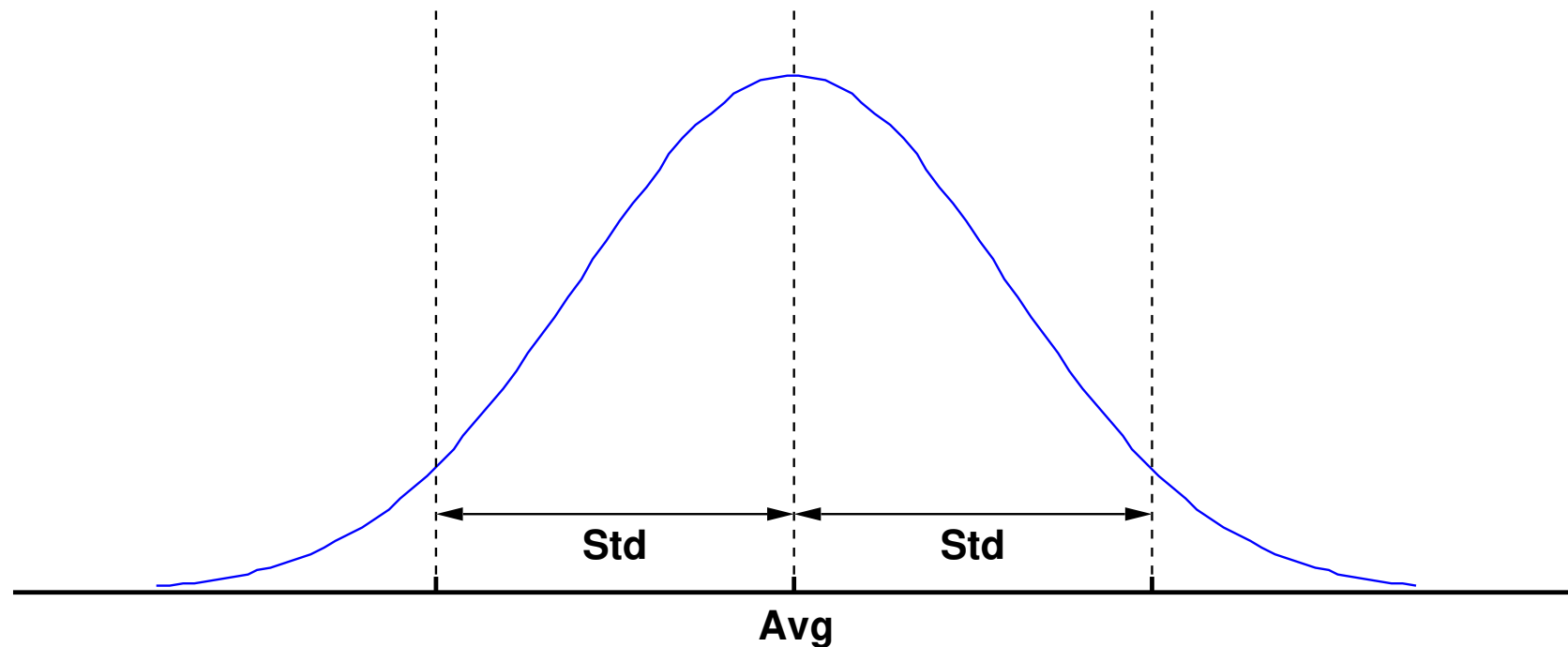
Grading

- ➡ Projects (i.e., programming assignments) graded by the graders
- ➡ Exams graded by the TAs
- ➡ Final letter grade assigned by the instructor
 - ➡ a grade of *incomplete* is only possible for *documented* illness or *documented* family emergency (according to USC policy)
 - please understand that it's not possible for me to get you a grade of *incomplete* because you need more time to improve your grade
- ➡ Two methods (assuming that no one games the system)
 - 1) on a curve
 - 2) fixed scale
 - ➡ your class letter grade will be the *higher* of the two
 - ➡ C's may be given, F's if necessary
 - pretty much the only way you would get an F in the class is if you *cheat*



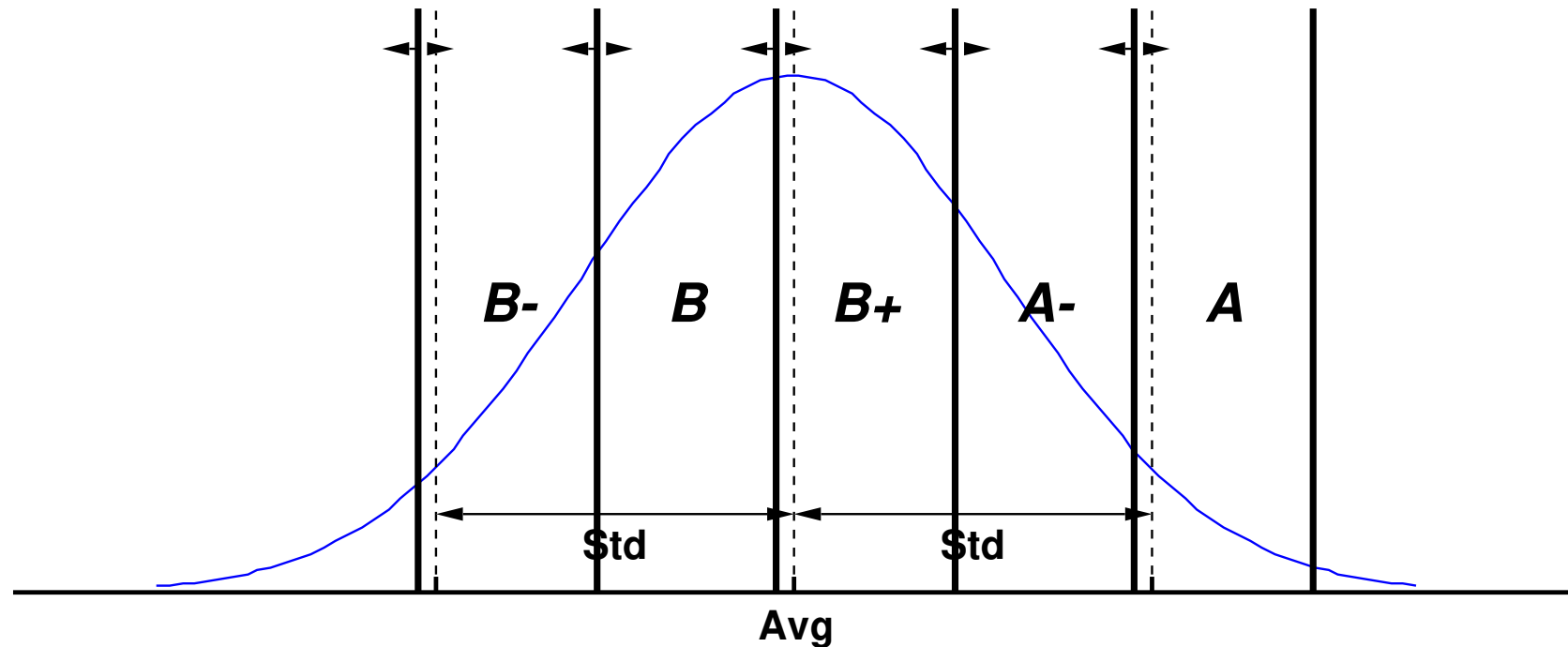
Grading

- ➡ **Curve (assuming that no one games the system)**
 - ▬ **one curve for each section (since exams are different)**



Grading

- ➡ Curve (assuming that no one games the system)
- one curve for each section (since exams are different)
 - loose guideline depicted below:



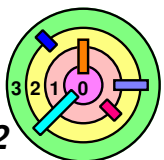
Grading



Fixed scale

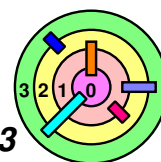
— every 9% is a "grade step"

Percentage	Letter Grade
91% or higher	A
82-91%	A-
73-82%	B+
64-73%	B
55-64%	B-
46-55%	C+
37-46%	C
28-37%	C-
19-28%	D+
10-19%	D
1-10%	D-
below 1%	F



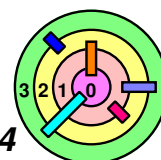
Academic Integrity Policy

- ➡ The USC Student Conduct Code *prohibits plagiarism*
- ➡ for warmup projects, all submitted work must be *your own work*
 - you must not *access/run/look at* code from previous semesters
 - you must not *copy* a single line of code from other sources
 - ➡ for kernel projects, all submitted work must be work done by members of your group
 - any member of your team must not *access/run/look at* code from previous semesters
 - any member your team must not *copy* a single line of code from other sources
- ➡ Two exceptions for copying code
- ➡ for *warmup assignments*, code fragments given to you as class resource in a class you have taken must be cited explicitly
 - must cite the code *inline* (or points will be deducted)
 - ➡ you can use any code *given to you* as part of *this class*
 - no need to cite such code



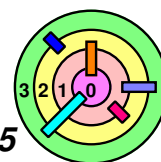
Academic Integrity Policy (Cont...)

- ➡ You are encouraged to work with other students for individual assignments or with other groups for group projects
 - "work with" does not mean "copy each other's work"
 - "work with" means discussing and solving problems together
 - this should happen at a *high level*
 - but be very careful when it's time to write code
 - must write code completely on your own
 - *do not write code together*
 - "sharing" even a *single* line of code is considered *cheating*
- ➡ If you cannot work together at a high level
 - you are advised not work together with other students for individual assignments or with other groups for group projects
- ➡ For more details, please see the *Academic Integrity Policy* section on the *Course Description* web page



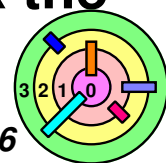
Academic Integrity Policy (Cont...)

- ➡ For kernel assignments, if you know your partner is cheating, you must tell him/her to stop
- there is only one submission from your team
 - you won't get graded separately from your teammates
 - if the university determined that the submission by your team was the result of plagiarism, everyone in the team will receive a grade of **F** for the class
 - even if you don't know that your partner is cheating, the same policy applies
 - choose your partners carefully
 - talk to your partners to make sure that no one is cheating
 - ◆ *throw away code written by other people*
 - ◆ if you know that a partner of yours has code from previous semester, you should ask him/her to throw away the code
 - talk to your partner to see if he/she can explain the code he/she claimed to have written



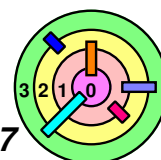
Academic Integrity Policy (Cont...)

- ➡ You must *not* post your code to a *public* code repository
 - ➡ if you post it to a *private* repository, you need to *verify* that it's truly private
 - ask your friend or kernel partners to verify
 - ➡ `github.com` is considered a *public repository* (no matter what they claim)
 - if you don't pay, your code becomes public
 - therefore, using `github.com` *is considered cheating*
 - `bitbucket.org` is free for students and you can setup private repositories
- ➡ Since our projects are on-going projects, you must not knowingly make your code public (not even pseudo-code)
 - ➡ if I find your code posted in a public place, you (and your team) will get a zero for your assignment
 - if you post your code after the semester is over, I will ask the university to change your grade



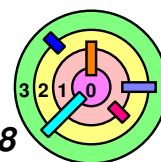
Displaying Your Code in Public Repositories

- ➡ As a general rule, you should only post code to the public *if the spec is public* and the code you are depending on is also public
 - all our assignment specs are *private*
 - the code given to you in our assignments are also private
 - *you must not post any of your CS 402 code to a public repository*
- ➡ You do not have the right to post it as part of your online resume
 - because your code depends on the rest of the assignment which you do *not* have the rights to *display* or *distribute*
 - it's clearly written at the beginning of every spec
 - this is serious business!
 - your future boss would/should appreciate that you understand about software copyrights
- ➡ Also, you have agreed to the USC Student Conduct Code
 - USC Student Conduct Code says that you must not cheat off other students and you *must not knowingly allow other students to cheat off of you*



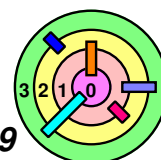
Program Checker

- ➡ Do not submit someone else's code
- ➡ How do we catch cheaters?
 - we use MOSS to analyze your submissions
 - <http://theory.stanford.edu/~aiken/moss/>
 - analyzes code structure intelligently
 - we have *all* projects from previous semesters
- ➡ If MOSS reports an unacceptable level of code match
 - your submission will be automatically rescinded and you will get a zero for that assignment
 - if you insist that we grade your assignment, I will have to forward the MOSS data to the university to investigate and decide if there is cheating before we can grade your assignment
 - if the university decides that plagiarism has occurred, you will get an *F* in the class
- ➡ I typically check for plagiarism around final exam time



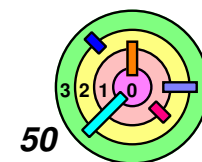
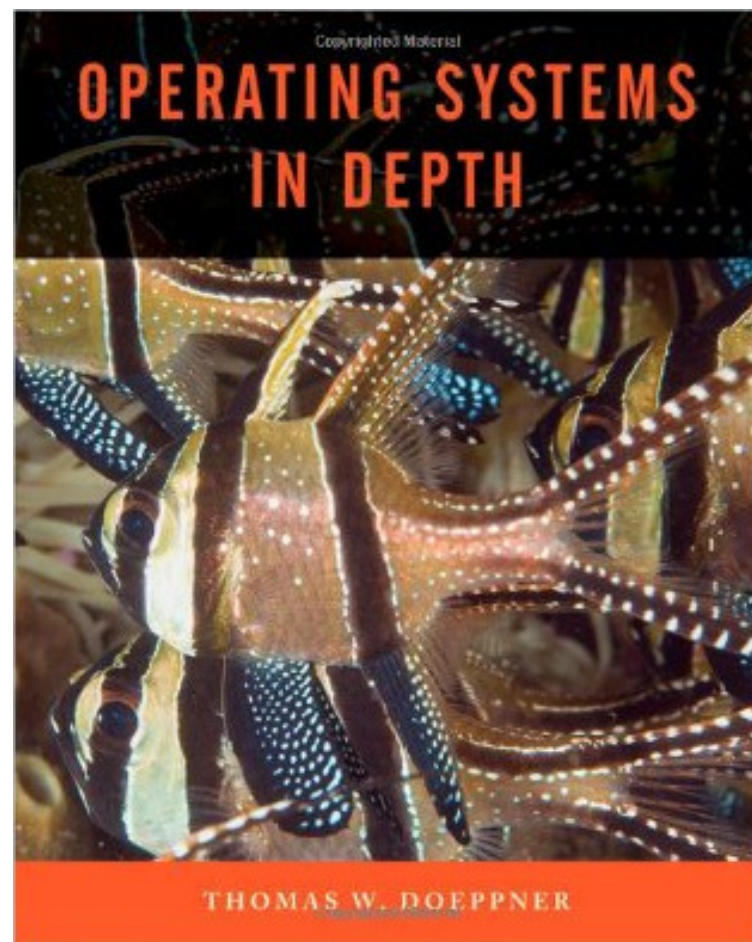
E-mail Questions

- ➡ One type of question I often get over email or see in class Google Group:
- here is my understanding of X. "Am I right?" "Is this correct?" "Correct me if I'm wrong..." "Please confirm that I'm right..."
 - this is really not a good way to ask something
 - ◆ if no one corrects you, you must *not* conclude that you were correct
 - stick to the definition of X in lecture slides or in the textbook and try to understand why it was stated that way
 - perfectly reasonable to discuss this during office hours
 - find a different way to ask *over email* to be more productive
- ➡ Another type of question I often get about assignments:
- I am thinking about not following the spec and do this instead. Is it acceptable (or is this okay)?
 - stick to the written words (spec and grading guidelines)!



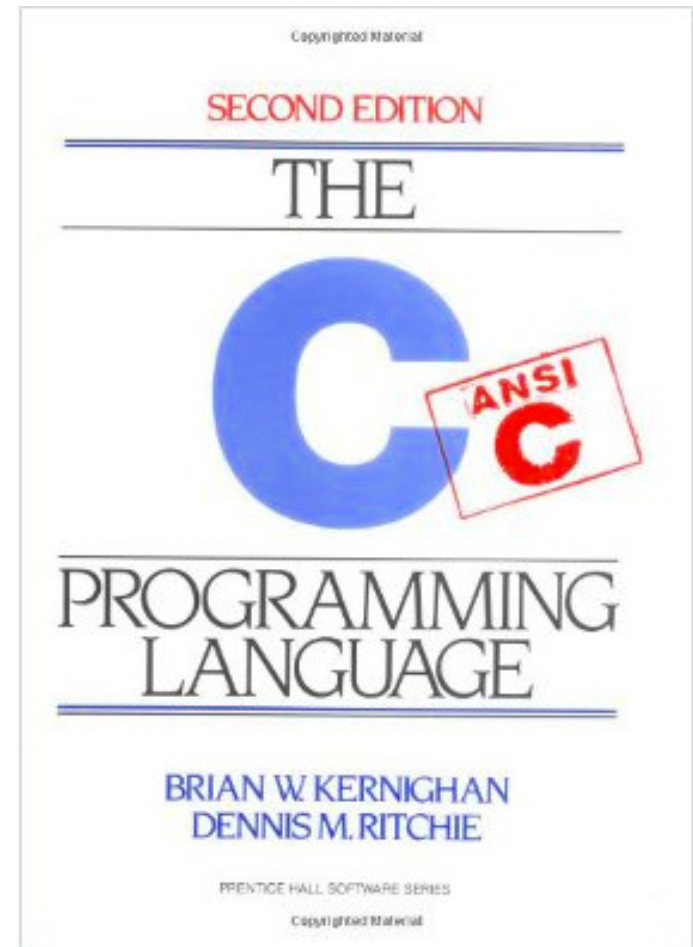
Course Readings

- ➡ **Required** textbook
- ➡ ***"Operating Systems In Depth: Design and Programming"*** by T. W. Doeppner
 - ➡ we will follow this book closely
 - my job is to explain this book to you
 - this book assumes that you have at least two years of solid computer science education
 - ◆ most students in CS 402 do not have such a background
 - ◆ other students in CS 402 did not take OS in undergrad
 - ◆ if you have a solide CS background and you have taken an undergrad OS class, you shouldn't be in this class!



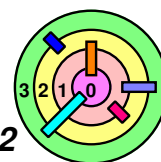
Course Readings

- ➡ **Optional** textbook
 - ➡ **"C Programming Language"**
by B. Kernighan and D. Ritchie
 - ➡ all programming assignments
must be done in C



Class Structure

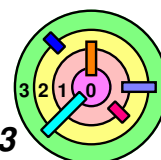
- ➡ Lectures mostly based on Doeppner
 - slides
- ➡ You must keep up with the assigned lecture videos and watch them before the corresponding live lectures
 - you are required to read the corresponding materials in the textbook
 - although exam questions *mainly* come from the lectures and slides
 - ◆ of course, "mainly" is not the same as "completely"
 - ◆ I can ask you about things that I think you should know, given what you are supposed to have learned in this class
- ➡ I expect you to *watch every lecture & discussion video on schedule*
 - the main reason why some students had a tough time with this class is because they simply didn't do that
 - you will also be expected to participate in midterm and final exams



Lecture Slides

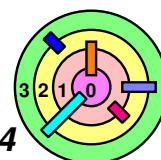
- ➡ **Lecture slides came with the book**
 - **the purpose of lectures is to explain what's in the textbook**
 - the textbook is not all that easy to understand
 - lecture slides are not meant as "extra material"
 - **lecture slides have a lot of details**
 - too much details?!
 - ◆ some are verbatim from the textbook
 - too much details can be a good thing
 - ◆ it tells you what's important to study
 - ◆ you don't need a separate study guide for exams!

- ➡ **You need to understand every slide**
 - **do you need to read the textbook?**
 - no if you just want to do well in the exams and you understand everything covered in lectures
 - although highly recommended
 - ◆ I think it's a good textbook



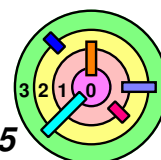
Quizzes

- ➡ To make sure that you are *keeping up* with pre-recorded lecture and discussion video schedule, we will have *quizzes* on some Fridays
- ➡ you will be quizzed on everything up to the current week with *focus* on the *lecture and discussion videos that hasn't been quizzed*
 - compare to exam questions, quiz questions will be "easier" questions, assuming that you have watched the videos
 - ➡ you download quiz questions (in PDF) and answers text file at the beginning of the quiz and you *upload the filled-out answers text file* before end of the day using a submission web form
 - ➡ grading details will be provided before the first quiz (rehearsal)
 - it will be the same as exam grading
 - ➡ no quiz on the week of the midterm or when a programming assignment is due
 - 8 quizzes planned



Quizzes

- ➡ Quizzes will be take-home, you download it at 3pm and you must submit *before midnight*
- starting at midnight, you lose 5% for every late minute
 - since it's a 9-hour quiz, you should be able to find time to take the quiz on Friday
 - no makeup and you cannot take the quiz at a different time since the answers will be posted soon after
 - we will automatically *drop two of your lowest quiz scores*
 - we cannot make exceptions
 - e.g., some students have Friday sailing classes and maybe out at sea without an Internet connection
 - ◆ this is a course time conflict with our discussion section and it's not approved
 - if this only happens twice in the semester, make sure you don't miss any of the other quizzes

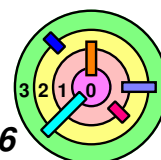


Projects / Programming Assignments



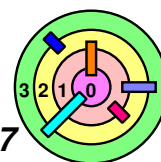
Programming assignments

- (7%, 18%) 2 warm-up projects (*to be done **individually***)
 - linked list in C, pthreads (no kernel code)
- (25%, 25%, 25%) 3 kernel projects (*to be done **individually** or in a **group** of 2-4 students*)
 - 1) kernel threads & processes
 - 2) virtual file system layer
 - 3) virtual memory (extremely difficult)
- **no solutions** will be given for any assignment
- program in C only (and you must learn C on your own)
 - C is a proper subset of C++
 - ◆ although stream I/O and strings are not available in C
 - ◆ you must learn how to do I/O the right way in C
 - ◆ you must learn how to deal with C-strings (i.e., null-terminated array of chars)
- the kernel assignments are **extremely difficult**
 - should get started as early as possible



Projects / Programming Assignments

- ➡ All assignments must be done on **32-bit Ubuntu 16.04**
- follow the instruction at the bottom of the class web page
 - do this as early as possible and let me know if there are problems
 - install VirtualBox first
 - then install a **virtual appliance** (which contains a 32-bit Ubuntu 16.04) into VirtualBox
 - ◆ we call this the **standard 32-bit Ubuntu 16.04 system**
 - ◆ we do not recommend that you "install from scratch"
 - ◆ if you want to use any other method, come **talk to me first**
 - if you are offered to "upgrade" to a new release, **you must refuse it**
 - ◆ or you have to **reinstall** Ubuntu 16.04 from scratch again
 - our kernel assignments will **NOT** work on Ubuntu 17.04 or newer
- ➡ You can install as many virtual machines as you'd like
- keep one **clean** standard 32-bit Ubuntu 16.04 for **testing** so you know exactly what the grader will see



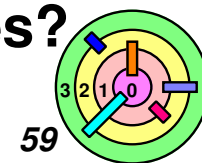
Projects / Programming Assignments

- ➡ It's a *really bad idea* to do a major upgrade of your host machine in the middle of the semester
- ➡ If you have a Mac with a non-Intel/AMD CPU, you won't be able to run Ubuntu 16.04 inside VirtualBox
 - ▬ you have 2 options
 - 1) borrow from family or friends an Intel-based Mac or a Windows machine to do our assignments
 - 2) run Ubuntu 16.04 inside a virtual machine on *AWS Free Tier*
 - *free for one year* but with limited functionalities (default settings are sufficient for this class) and *usage caps*
 - ◆ if you go over usage caps, they will charge your credit card
 - if you only use this account to do our assignments, most likely, you will not go over usage caps
 - if you are getting close to your usage caps, just create and use a new AWS account
 - ◆ super important to *backup your work*



Projects / Programming Assignments Grading

- ➡ For *kernel assignments*, 25% the grade is "team grade" and 75% is "grade based on individual contribution"
- ➡ if your teammate is a *free-loader*, that teammate will still get 25% of your grade (good reason to choose your teammate carefully)
 - ➡ if every team member contributes *equally*, everyone will get the *same grade*
 - ➡ in the README file your team will submit, you need to:
 - specify how to split the points
 - ◆ in terms of percentages (or fraction) and must sum to 100% (or 1)
 - give a brief justification about the split if it's not an equal split
 - ➡ for example, the grader gives you 80 points and everyone contributed equally, everyone gets 80 points
 - ➡ what if it's not equal contribution?
 - what if the split is 0/50/50? what should be your scores?
 - what if the split is 20/30/50? what should be your scores?



Projects / Programming Assignments Grading

➡ If there is an uneven split, we will use a program called "cs402calc.txt" (it's a perl script) to assign scores:

— general syntax

```
cs402calc.txt grade p1 p2 ...
```

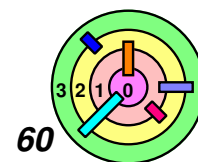
○ where `grade` is the grade the grader gave, and `p1`, `p2`, ... are percentages

— simple linear interpolation

○ Ex:

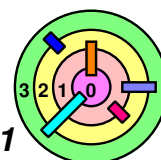
```
cs402calc.txt 100 70 10 10 10
```

- ◆ scores would be 100, 35.71, 35.71, and 35.71, respectively
- students who contributed 0% would still get a 25% "free ride"
- if you take on a teammate, you implicitly agree to accept both risks and rewards
 - ◆ if risk is unacceptable to you, you should do the kernel assignments solo



Projects / Programming Assignments Grading

- ➡ It can be difficult to figure out contribution of each team member
 - some would try to do that precisely and that can create tension among team members
 - is finding a bug worth writing 100 lines of code?
 - how much less important is testing compared to coding?
 - I cannot and will not get involved in your team's dispute because I did not get involved in forming *your team*
- ➡ My recommendation is to simply *claim* equal contribution in your README file and focus on getting the work done
 - what about free loaders?
 - when you form your team, *you need to have a discussion* with your teammates and agree on how to determine the split

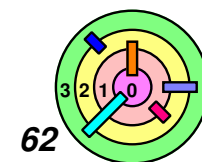


Projects / Programming Assignments



Forming groups

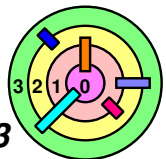
- you can form a group with students from *any* section
- It's probably a good idea to work with other students during the warmup projects
 - although you must *write code independently* for the warmup projects
 - "work with" means working at a high level (*not* code level)
- all students not belonging to a group by the group forming deadline will be assigned a group
 - algorithm:
 - ◆ form a random list from these students (random drawing)
 - ◆ assign 4 to a group starting from the beginning of the list
 - ◆ remaining students join these randomly formed groups
 - ◆ this is the only way to have 5-student groups
 - ◆ boundary conditions? hope we don't get there
- after groups are formed, only mutually agreed swaps are allowed, and with my approval



Projects / Programming Assignments

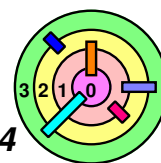
- ➡ We know that our algorithm is far from perfect
 - that's why it's specified way in advance
 - this way, you can plan how to form teams given that you know our algorithm for assigning teams

- ➡ If you cannot form a team of your own and end up in a team that cannot get the kernel assignments to work?
 - probably because there are problem members
 - is this unfair to you?
 - No! because you did have options
 - given that you know that this is the rule, what should you do?
 - form your own team, or
 - do the kernel projects by yourself!
 - if you *choose* neither, we cannot grade you differently *given our fairness policy*
 - during the first 6 weeks, work with other students so you know with whom you want to be partners



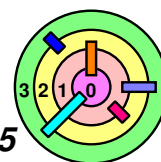
Grade Normalization for Programming Assignments

- ➡ It's pretty much impossible for graders to grade identically
 - students in different sections can be in the same kernel team makes things more complicated
 - if your team gets a score of 90, it may be above average for one student and below average for another
 - if another team scores 90 as well, it may be graded by an easier/harsher grader
- ➡ When I calculate your final grade, I will *normalize* each of your *programming assignment grades*, using the average of your grader
 - e.g., average of your grader is 85 and your score is 87.5
 - your score is thus "average plus 2.5 points"
 - your normalized score will be the overall class average plus 2.5 points
 - e.g., if your score is "average minus 2.5 points", your normalized score will be the overall class average minus 2.5 points
 - I know this is not perfect, but it's much better than using un-normalized scores



Electronic Submissions

- ➡ Use the *Bistro* system (see bottom of assignment specs for details)
 - you can make multiple submissions
 - will grade last submission by default
 - Bistro system gives *tickets* for your submission
 - these are *proofs* that my *server* got your submission
 - *we cannot trust any file timestamp*
 - we can only trust things that have made it to our server
 - very important: *read your tickets and save your tickets as PDF*
 - very important: *verify your ticket and verify your submission*
 - see the bottom of the Electronic Submission Guidelines web page for details
 - if you forget a file in your submission, you are not allowed to resubmit it after the deadline
 - submit source code only (or 2 points will be deducted)
 - for team projects, only one member needs to submit
 - it is *your responsibility* to make sure that your submission is what you want us to grade - *Be a little paranoid!*

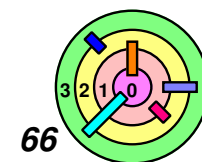


Electronic Submissions

- ➡ Use the *Bistro* system (see bottom of assignment specs for details)
 - no other form of submission will be accepted
 - use your judgement under special circumstances or urgent situation while *accepting the risks* with email submissions
 - ◆ please understand the nature of emails, i.e., it can take *hours* or even *days* for an email to get delivered (and your submission will be considered late)
- ➡ *Verify* what you have submitted
 - if your submission file is "warmup1.tar.gz", the hash_value in the *ticket* should match the printout of the following command:


```
openssl sha1 warmup1.tar.gz
```

 - SHA1 hash value of a file can be considered as a *digital fingerprint* of that file



Late Policy



Electronically Submitted Assignments

- you can submit multiple times, only the last submission will be graded (unless you send the instructor an email)
- 14 minutes and 59 seconds grace period
- at 12:00:00am, you will start losing 1% of your grade every 5 minutes (round up to the nearest percent)
- see next page for details
- time is based on *Bistro server timestamp* in the *ticket*
 - you need to be mindful about the difference between your clock and the clock on the Bistro server

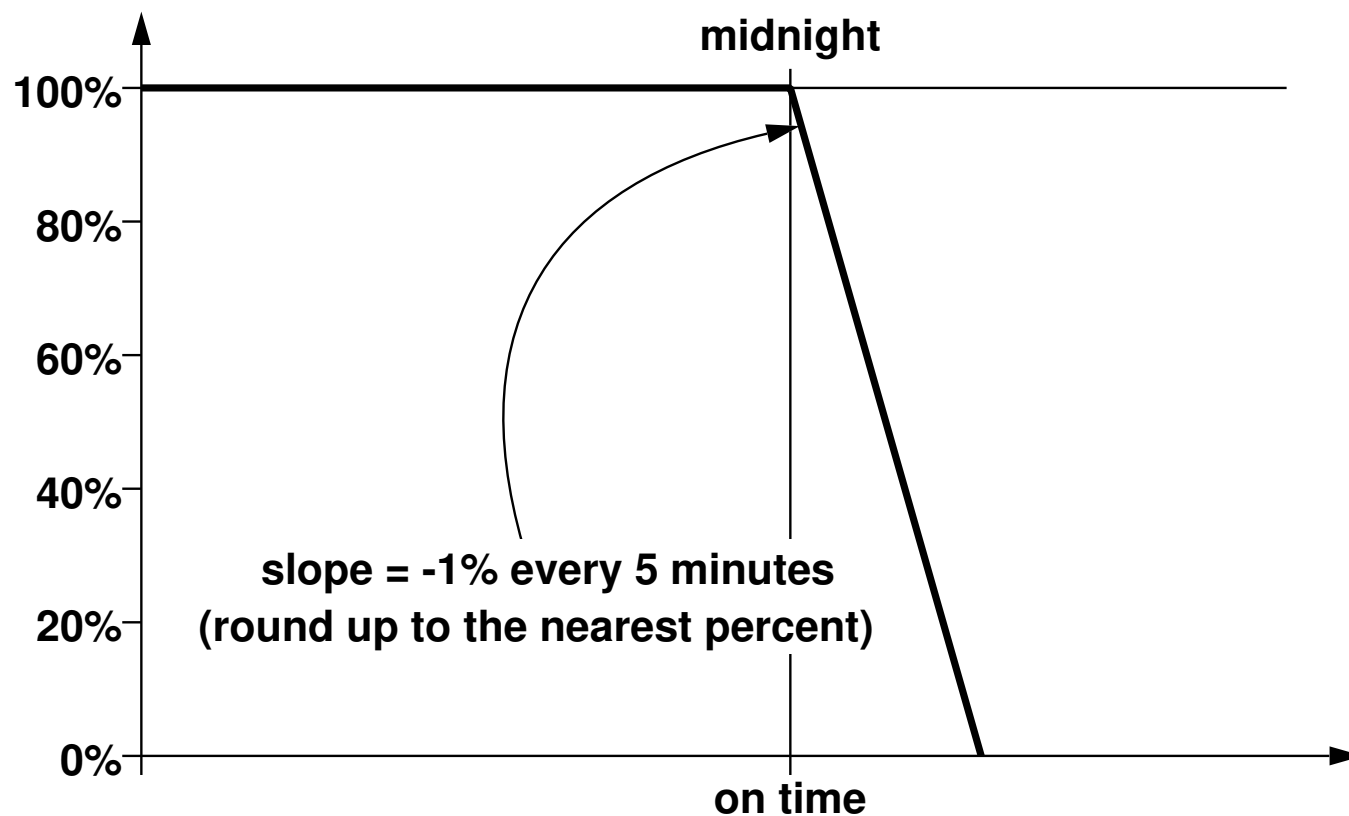


Extension *only possible* if you have a "*note from a doctor*" or other form of official proof of family emergencies

- "note from a doctor" just means something that shows that you were at the doctor's office or student health facility
- e.g., scheduling conflict with work, other classes, etc. cannot be excused

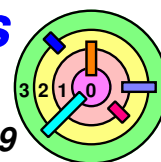


Late Policy



Late Policy

- ➡ Each student also have 2 "free late days" that can be applied to warmup assignments and 3 "free late days" that can be applied to kernel assignments
 - you decide how you want to distribute these "free late days"
 - you can put them all on one assignment or spread them around
 - although for kernel 3, you can use at most 1 free late day
 - for a kernel assignment, a "free late day" will cost each team member a "free late day"
 - if you have used a "free late day" on an assignment, *once that assignment is graded*, you cannot "un-use" the "free late day"
- ➡ A "free late day" can only be used on a *late submission*
- ➡ For all assignments (except for kernel3), you can submit before the next assignment deadline for a 50% deduction
 - requirement: you must inform the instructor *within 72 hours of the assignment submission deadline*



Late Policy



I must stick to my policies

1) please do not ask for individual extension unless you have a *documented* proof of *illness* or a *documented* proof of *family (not personal) emergency*

2) my "fairness" policy is: *"Whatever I offer you, I must offer it to the whole class"*

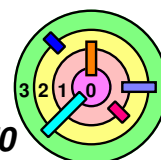
◆ this is why I cannot give individual extensions

— what if your laptop died? home Internet disrupted? car broken? cousin got stuck at the airport? need to go to court? need to go to Miami? and so on ...

○ these are personal emergencies

○ please see (1) and (2) above

○ best to *submit early* so personal emergencies will not cause problems

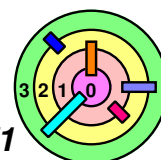


Modifications After Deadline



After the submission deadline has past

- you are allowed up to **3 lines of free changes**, if submitted via email to the **instructor**, **within 24 hours** of the project submission deadline
 - clearly, this is not meant for major changes
 - you may want to anticipate that your submission may not be exactly what you thought you had submitted
- one line (128 characters max) of change is defined as one of the following:
 - **add** 1 line before (or after) line x
 - **delete** line x
 - **replace** line x by 1 line
 - **move** line x before (or after) line y
- additional modifications at 3 points per line **within 24 hours** of the project submission deadline

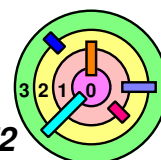


Modifications After Deadline (cont...)



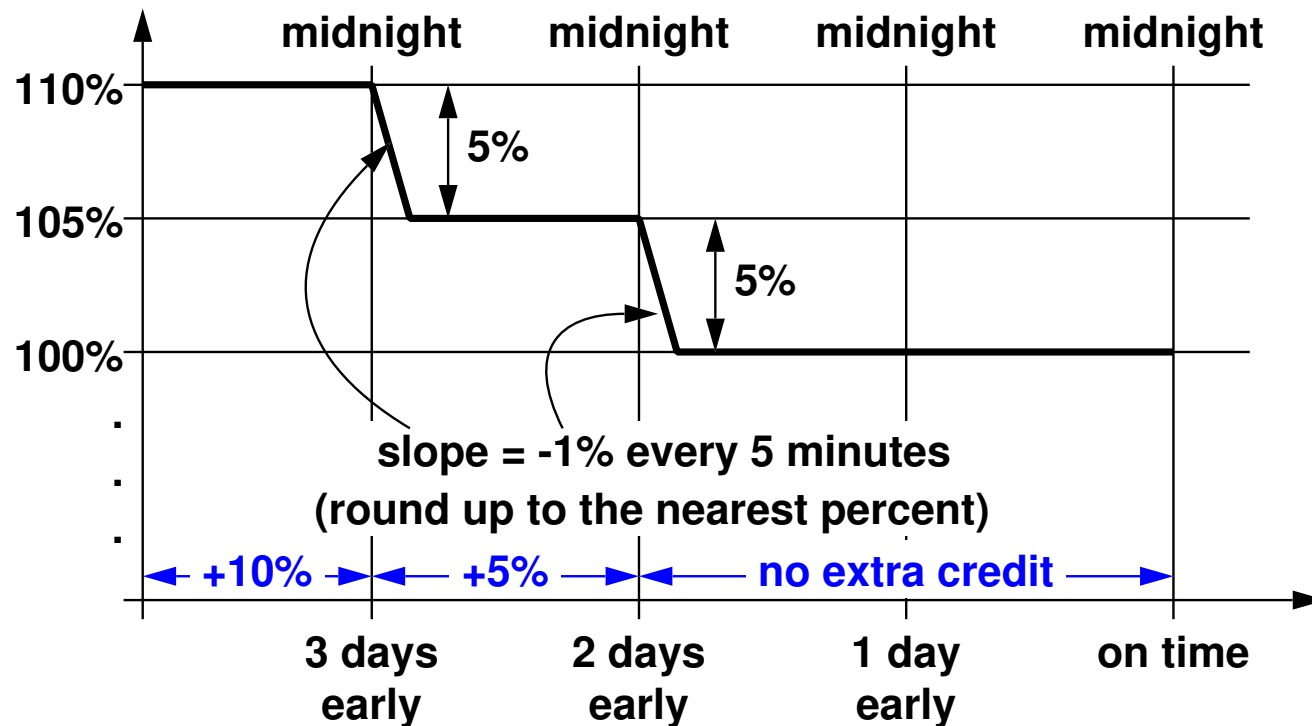
After the submission deadline has past (cont...)

- 24 hours after the submission deadline, additional modifications cost **10 points per line** for the next 6 days
- afterwards, it costs **25 points per line**
- applies to source code and README files
 - do not forget to submit files, and **verify your submission**
 - ◆ this is **your responsibility**
 - we cannot accept missing files after deadline because that's too many lines of changes!
 - a **file system timestamp** can be easily **forged**, so they cannot be used as **proof** that you have not modified the file after deadline
- try things out before your first submission deadline to get familiar with the **Bistro** system
- **re-test** your code after you have made your final submitted to be sure



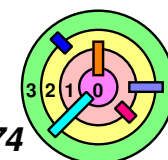
Extra Credit 1

- ➡ To encourage you to do your projects early, you will get extra credit if you turn in programming assignment 2 or 3 days early
- ➡ if your submission is more than 72 hours before the posted deadline, you get an **extra 10%**
 - ➡ if your submission is between 48 and 72 hours before the posted deadline, you get an **extra 5%**



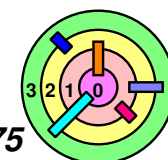
Extra Credit 2

- ➡ You can get extra credit for posting *timely*, *useful*, and *insightful* answers to the class Google Group in response to questions posted by other students regarding *kernel programming assignments*
- ➡ you probably won't get extra credit if you repeat exactly what another student has already posted
 - you need to *add* something useful to existing posts
 - you can still post to support another student's position
 - ➡ you probably won't get extra credit if you respond more than *12 hours* after the time of the original post
 - ➡ the maximum number of extra credit points you can get is *10* points for each of the kernel assignments (on a 100-point scale)
 - ➡ if you post something good, it's your responsibility to verify that it has been posted to the Google Group
 - ➡ you can *lose* extra points you have earned if you post something unprofessional to the class Google Group or exhibit poor netiquette



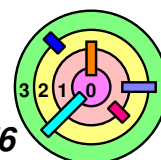
Regrade Policy

- ➡ Grades will be sent to individuals via email to your USC email address
 - setup filter to not miss emails from <bill.cheng@usc.edu>
- ➡ Regrade requests in writing
 - submit within 1 week of initial grade notification
 - must follow instruction in grade notification email
 - regrade can be done after the 1-week deadline, but you must *initiate* a regrade request within 1 week
 - we reserve the right to regrade the whole thing
- ➡ If you have made multiple submissions and after you have received your grade, you realized that you meant for us to grade a different submission
 - it will cost you 20 points to *regrade a different submission* at this point
 - this is why *Verify Your Submission* is so important to make sure that we are grading the correct submission

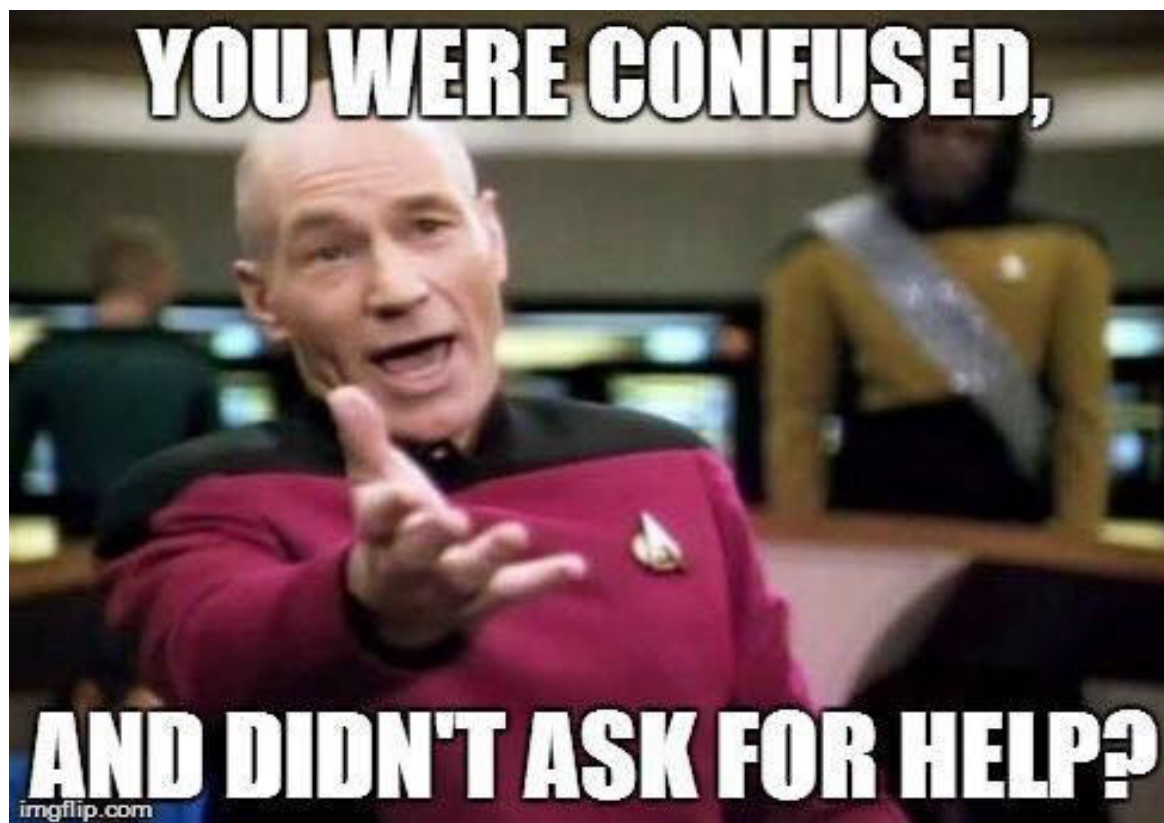


Student Commitments

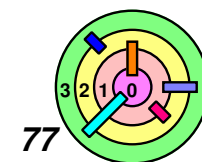
- ➡ Keep up with *lectures* and *discussions* (*live and pre-recorded*)
- ➡ Do your own work
- ➡ Turn in assignments on time
- ➡ Ensure gradable assignments
 - save a copy of the *ticket* you see in the browser
 - it's a *PROOF* that my server has received your submission
 - the *ticket* in the email is not that important and can get delayed
 - it's super important to *verify your submissions*, especially after you have made your *final submission*
 - if you submitted a wrong file or forgot to include a file, there is absolutely nothing we can do after the deadline
- ➡ You are encouraged to study with other students and *discuss* (no copying/sharing code or pseudo-code) about programming assignments



Student Commitments

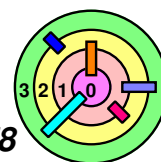


- ➡ If you are confused about anything, you are *expected* to ask me questions!
- you are encouraged to ask questions (not getting tutored), pretty much about anything related to programming in C
 - when you get stuck with programming, ask the TA or the instructor for help, don't wait too long



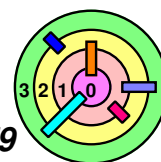
Student Commitments (Cont...)

- ➡ If you feel that you are falling behind
 - talk to the instructor as soon as possible
- ➡ How do you know you are in trouble?
 - assuming that you want a decent grade
 - you look at a set of slides that has been covered in class and have no idea what it means
 - then you read the textbook and understands it perfectly
 - ◆ you are probably not in trouble
 - if you read the textbook and are still confused
 - ◆ you are somewhat in trouble
 - if you don't read the textbook
 - ◆ you are probably in *big trouble*
 - don't do this right before exams!
 - you should ask yourself, "Am I in trouble?" like at least every week if you don't plan to come to lectures



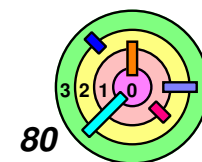
Study for Exams

- ➡ Exams are worth a lot! This class is not just about programming!
 - how many kernel programmers do we need in this world?
 - but everyone in CS must understand *OS concepts* well
- ➡ Exams mostly are based on lectures
 - I reserve the right to ask anything from required materials
- ➡ If you want to good grade in this class, you have to do well in the exams and assignments
 - just do well in the assignments is not enough
 - to do well, you have to *study* to understand the materials well and be able to *think clearly* about the concepts learned
- ➡ Do not review all lectures only right before the exams
 - otherwise, you may only be able to give generic answers because everything is a blur
 - you need to show that you know the difference between answers of different quality



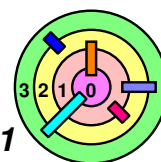
Too Much Time Programming

- ➡ Some students complained that they had to spend too much time programming and debugging
- that's really not supposed to happen
 - it's mainly because students *didn't understand what they are doing with their code*
 - it's important to understand what's covered in lecture and discussion videos because they provide you with background
 - don't just hack your code till it works (because it won't work)
 - for every line of code you are writing, think about the following (clearly not an exhaustive list)
 - what have I assumed before I execute this line of code (or function)?
 - what's the effect after I execute this line of code (or function)?
 - can I overflow a buffer with this line of code (or function call)?
 - ◆ if you copy data into a buffer, is the buffer big enough?
 - ◆ if you are using an array index, are you sure that the index is always valid?



Too Much Time Programming

- ➡ Some students complained that they had to spend too much time programming and debugging (cont...)
- memory corruption bugs are nasty and difficult to debug
 - best is to *write your code very carefully* and make sure that you don't have memory corruption bugs
 - remember, the bugs you see are *your bugs*
 - *you* put it there!
 - much better to *avoid creating bugs* than to *debug*
 - ◆ this can save you a lot of time
 - ◆ you need to learn to be disciplined and thoughtful when you write your code in the first place



Things to Do *Today*



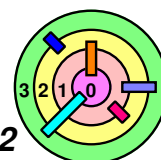
Read entire class website

- get *username* and *password* to access protected part of website
 - should do this even if you are not registered for the class but plan to take this class
- check warmup #1 project spec and *start coding*



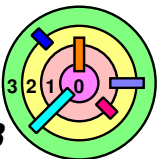
Additional things to learn/do quickly

- learn C if you don't know it already
- learn "git" (see online book)
 - you will need it for group projects
 - should start using it for the warmup projects
- learn a commandline editor: `vim/pico/emacs`
 - `pico` is the easiest
- install VirtualBox on your laptop/desktop and then install *standard 32-bit Ubuntu 16.04* into VirtualBox and let me know if there are problems
 - if the only machine you have is a M1 Mac, contact me!



Summary Of Zoom Meeting IDs

- ➡ Due to security concerns, we cannot post Zoom meeting IDs in the public area of our website
 - go to class home page, click on Videos, scroll all the way to the bottom and click on Zoom Meeting IDs
- ➡ **USC authentication** is **required** for **all Zoom meetings**
 - no passcode and no waiting room
 - if you somehow end up in a waiting room, it must mean that you were using Zoom authentication (and did not use USC authentication) and I will not admit you

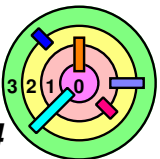


Course Content Credit

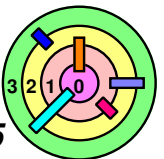
- ➡ Slides and course content primarily came with the textbook and originally written by:
 - Prof. Tomas W. Doeppner
- ➡ Additional slides and course content from:
 - Prof. Ramesh Govindan
- ➡ Some (test) code for the kernel assignments from:
 - Prof. Ted Faber
 - Dr. Sung-Han Lin

Miscellaneous

- ➡ Someone actually complained about this, so I'm stating it here
 - I do not use the standard course evaluation window
 - in the old days, course evaluations are done in class (i.e., the window was about 15 minutes)



Extra Slides



Going Into "Secondary Live Lecture"...

- ➡ Live and in-person lectures
 - = **(DEN section)** MW 10:00am - 11:20am in OHE 136
 - includes section **29945D** (local DEN) and section **29946D** (remote DEN)
 - they are the same section because they have identical class meeting time
 - ◇ therefore, whenever I use the term "**DEN section**", I am referring to section 29945D **and** section 29946D
 - = **(TT section)** TuTh 9:30am - 10:50am in WPH B27 (section **30331D**)
- ➡ Midterm: during class time, Wed, 10/30/2024 (*firm*)
- ➡ Final: **(DEN section)** 8am-10am, Mon, 5/6/2024 (*firm*)
(TT section) 8am-10am, Tue, 5/7/2024 (*firm*)

Going Into "Secondary Live Lecture"...

- ➡ I'm going to start the *secondary live lecture* on Zoom soon
 - ➡ go to class home page and click on Videos, scroll all the way to the bottom and click on the link for Zoom meeting IDs
 - make sure you have signed into Zoom (<http://usc.zoom.us>) using *USC SSO authentication*
 - you can watch remaining lecture videos during this meeting
 - feel free to ask me questions during this "live lecture"
- ➡ Starting with next class, "primary live lecture" will mostly be *short*
 - ➡ I will give *administrative information*, do a *quick recap* of what's in the recorded videos, and ask if you have *questions*
 - ➡ when there are no questions, I will go to my office to start the "secondary live lecture" to continue the live class, all the way to the end of the live class

