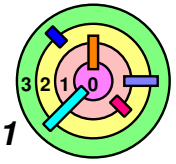
CS 350
Introduction to Operating Systems

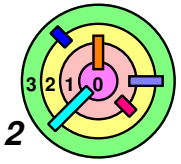
Bill Cheng

http://merlot.usc.edu/william/usc/



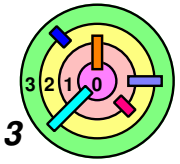
About This Class

- This is a junior/senior level undergrad class covering the *fundamental concepts* of operating systems
  - this is a foundation class and not a "tech class"
    - it is not about the latest products and functionalities
    - this course is meant to *provide background* so you can *understand* the designs, algorithms, and techniques used in current operating system products and functionalities
- You will also *learn to write OS kernel code*
  - there is a CS 402 class on OS, but that's for grad students
    - different textbooks, different programming assignments, and different approaches to programming assignments,
    - lots more code to write in CS 402 programming assignments
  - undergrad students must take CS 350 for OS credit



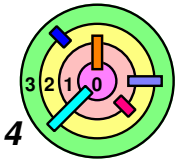
Today's Topics

- Administrative Stuff
- Review Course Organization
  - please read all the administrative information and lecture slides
- Please pay special attention to today's lecture
  - I am a stickler to *rules*, especially *written rules*
    - the grading rules are setup to keep things simple and keep things fair
  - I don't want you to get a zero in some assignments because you are not aware of how serious grading rules must be taken (often time, literally)



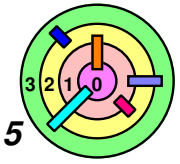
The Importance of "Written Words"

- We communicate (here and in the real world) using "words"
  - spoken words may be unreliable
    - no matter how hard I try, I mis-spoke sometimes
      - I can't slow down much because there is too much to cover
  - "written words" are different
    - you need to take written words seriously
    - especially when it comes to *rules written in words*
      - if we don't take them seriously, why write them down?!
  - things that are not written down can get messy
- If it's writtent that X is the only grading procedure and that we must follow the grading procedure
  - what would we do if you ask us to grade your submission using a different procedure?
    - we won't, because we take *written words* very seriously
- Words on lecture slides are important even if I skip them (because they are written words)



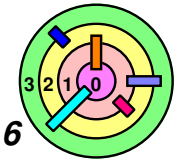
The Importance of "Written Words"

- When it comes to exams, we can only grade based on what you *wrote* on the exam paper (and not what's in your mind)
  - you need to learn to *choose your words* carefully
- Please pay attention to *all* the written words anywhere in the *class web site*, *lecture slides*, posting to class Piazza by *me*
  - setup your email filter to not miss messages from <bill.cheng@usc.edu>



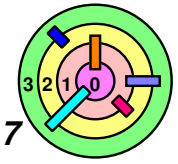
Fairness

- Without *fairness*, grades have little meaning
  - I am required to give you a grade
  - the instructor *must treat all students equally* and cannot give special treatment to any particular student
  - therefore, please do not ask special favors from the instructor because of your circumstances (except for ones that are explicitly allowed by the university)
  - this may seem unfair to you because you believe that your circumstances are special (understandably, everyone does)
  - bottom line, the rule the instructor must follow is that *whatever he offers you, he must offer to the entire class*
    - other than the exceptions that are setup at the beginning of the semester (mostly for students with university-approved accommodations)
    - we definitely will not grade based on effort *(i.e., there will be no partial credit for just "effort")*
      - you get partial credit for *passing tests*



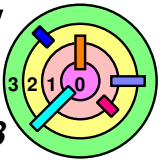
I Must Stick To All My Written Rules

- Some students don't understand why I'm so strict with my rules
  - it's not a power trip for me
  - I am *bound by my own rules*
    - rules *take away power* from me
- I'm responsible of treating all students fairly
- If I apply one rule for one student and don't apply the same rule for another student, that's *totally unfair*
- The only way I know to be fair to all is to stick to all my written rules
- When you ask me to bend a written rule for you, you are asking me to be unfair to other students
  - therefore, I will not bend a rule for you



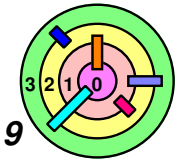
Class Structure & Teaching Staff

- Instructor: Bill Cheng <bill.cheng@usc.edu>
  - email policy: 24 hour turn-around
    - this is my *promise* to you (but applies only to *private* emails)!
  - office hours: on Zoom, Tu/Th 9pm-10pm or by appointments
    - these Zoom meetings *will not be recorded*
    - maximum meeting time for a student will be 15 minutes
    - individual meeting in a break-out Zoom room where you can share your screen if you'd like
      - everyone else wait in the main room, first-come-first-served (you should *raise hand* immediately so that Zoom can put you in a queue)
      - students in other classes I teach may also come to my office hours
      - if there is a lot of students waiting, you may not end up with a meeting with me
    - if your questions can be handled over email, it's probably best to send me email, or post to Piazza



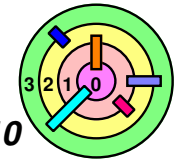
Appointments

- I would grant appointments with me only under very special circumstances and *only for personal/private administrative matters*
  - if such matters can be handled in a break-out room on Zoom during office hours (which is a private space), that would be preferred
  - if you want to talk to me about course materials or programming assignments, please come to office hours, send me email, or post in Piazza
    - using appointments to get extra help on assignments would be unfair to other students (unless I accept appointments whenever I'm asked, and that's usually not feasible)



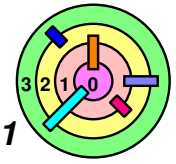
Office Hours

- Please understand that office hours are for me to *help* you, not to "tutor" you
  - same goes for the TA's office hours
  - please understand that *we are not your paid tutors*
    - if you are missing background on something, please ask me and I can point you to the right places to get information
    - you are expected to read Linux "man pages" and online documentation to learn how to use new commands and functions
  - Viterbi have tutoring services (may not be available for CS 350, but you can make a request)



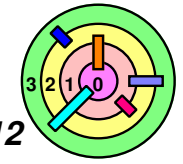
Class Structure & Teaching Staff

- **TAs:**
  - we have 6 TAs this summer
    - their office hours will be posted soon
  - email: 24 hour turn-around
  - a TA's main job is to *help* you with the course materials and programming assignments
    - TA cannot *do work for you* (such as *find bugs* in your code, *write code* for you to use, etc.)
    - TA *cannot tell you what code to write*
    - TA *cannot look over your code* or *comment on your code*
    - it's not the TA's job to "tutor" you
    - TA can help you with learning to use the debugger
- You can go to *any* TA for help with course materials and programming assignments



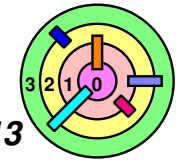
Class Structure & Teaching Staff

- Graders and CPs: none this summer since we have so many TAs
  - the TAs will grade programming assignments
  - please don't contact the TA to ask, "Here's my assignment. How many points will I get""
    - please just follow the grading guidelines and pretend that you are a grader (because that's what the TA must do to grade)



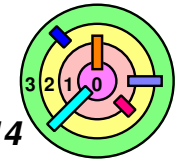
Office Hours

- Office hours are for answering questions
  - they are not intended for finding bugs in your code
    - especially since each student can only have 15 minutes
  - *finding bugs is your job* and it's an important that you *acquire such a skill*
  - if you need debugging help, we can only give you suggestions and tell you what `gdb` commands to try
- Please understand that *we don't know where your bugs are*
  - some bugs are so difficult to find, even for professionals
  - therefore, please do not expect that we can find your bugs
- Some students think that we are holding out on them, that we know where the bugs are and we are just not telling them
  - that may be the case when the bugs are trivial and we want you to learn to spot the bugs
  - often times, we just don't know where your bugs are (there are just too many possibilities)



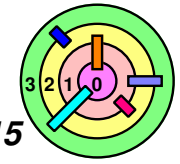
Office Hours On Zoom

- If no one comes to an office hour on Zoom, we may walk away from Zoom temporarily (but we will not be far)
  - we will put a note on the screen to say that we are away momentarily
  - call out our name or send us email to ask us to come back to Zoom
  - if you don't see a note on the screen, it's most likely because we are in a break-out room and we may not know that you are waiting (you can send us an email in case there is an operator error)
- If an office hour is moved or canceled, I will add a news item in the News section of the class home page
  - if an office hour Zoom meeting is not active, you should check the class home page to see if it has been moved or canceled
    - also, send email



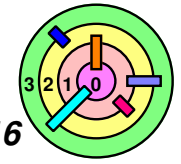
Class Resources

- **Class web page: http://merlot.usc.edu/cs350-m25**
  - **everything about this class is there**
    - **anything related to *grading*, you are *required to know***
  - **lectures and assignments are *password protected***
  - **get familiar with it**
    - **if you are not used to reading a lot of stuff, you should start reading a lot of stuff in this class**
      - **it's important to learn how to *read* documents and *interpret* them *correctly***



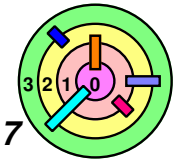
Class Resources

- If you see ***inconsistencies***, especially regarding any type of "rules" between what's on the class web page, what's on a set of lecture slides that has been covered in class, or what I said
  - usually, the lecture slides are correct
  - but you should ask me which is correct as soon as possible
    - please understand that I do mis-speak often enough (and my English is not perfect)
      - it's more important that ***written words*** are consistent
    - I will address the inconsistency and make changes so that things can be consistent again



Lectures

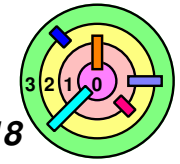
- The lecture slides will mainly follow the textbook
  - some would say that this is boring
  - our textbook is used by many CS departments in the US and it's important to go over the materials there in enough detail so you don't miss things that you are supposed to know
- Why not just read the textbook and not come to lectures?
  - *most* exam questions will be *based on* lectures and lecture slides
    - some stuff in lectures are *not* in the textbook
    - exams are about *your knowledge of this class*
      - exams are *not* about your *general knowledge of OS*
  - it's important that you *understand* the lecture slides well enough so you can give correct answers for trickier exam questions
    - you should also understand the corresponding material in the textbook, when applicable
  - you can use the lecture slides as a study guide
    - there's a lot of details in the slides



Class Discussion Forum (Piazza)

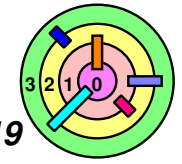
- Piazza Forum
  - student-to-student discussions about labs, programming assignments, and course material
  - exchanging *ideas* are allowed
  - posting code is *not* allowed (short code segments to illustrate ideas are allowed; short means < 3 lines *and* < 3 function calls)
    - *first offense* (in the entire semester) gets a warning (unless it's a lot more than the threadhold)
    - *2nd offense*, you will lose 50% of the corresponding assignment points and lose posting privileges
  - instructor and TAs will also post answers to questions here
    - if appropriate for whole class

- You *must be a member* of the *class Piazza Forum*
  - *all* important announcements will be posted to this group
  - I will add you to the Piazza Forum after Lecture 1



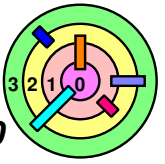
Grading

- **Programming** 30% 5 kernel programming assignments
- **Midterm** 35% in class, Thu, 6/26/2025 *(firm)*
- **Final** 35% in class, Tue, 8/5/2025 *(firm)*



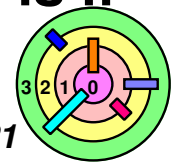
Grading

- Additional extra credit
  - turn in *PAs* more than 48 hours before deadline
  - 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!



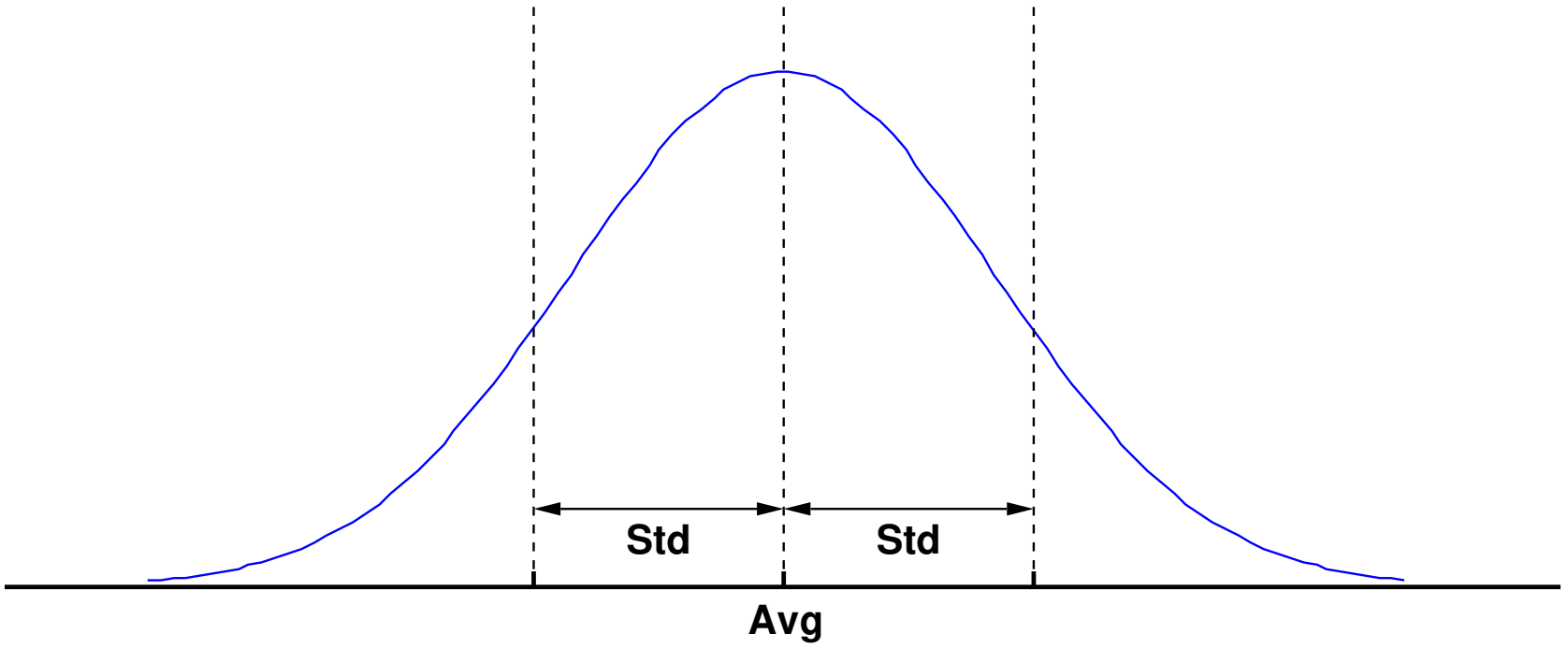
Grading

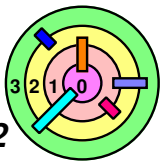
- Programming assigments graded by grader
- Exams graded by the instructor
- Final letter grade assigned by the instructor
  - a grade of *incomplete* is only possible for *documented* illness or *documented* family emergency (according to USC policy)
    - please understand that it's not possible for 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
  - D'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

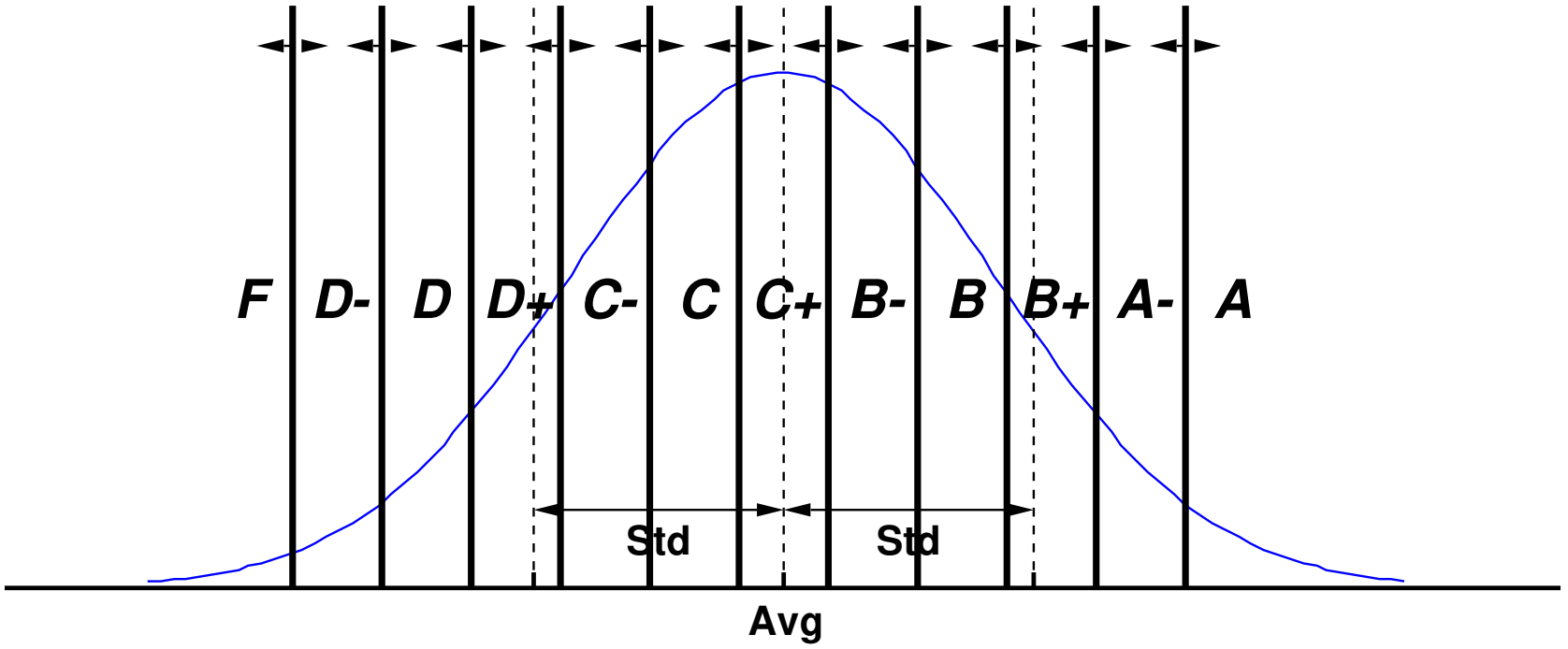
- Curve (assuming that no one games the system)
  - compute weighted sum and plot on a curve





Grading

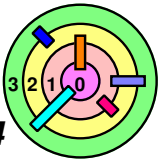
- Curve (assuming that no one games the system)
  - compute weighted sum and plot on a curve
  - loose guideline depicted below:



Grading

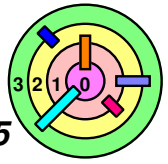
- **Fixed scale**
  - **every 6% is a "grade step"**

Percentage	Letter Grade
94% or higher	A
88-94%	A-
82-88%	B+
76-82%	B
70-76%	B-
64-70%	C+
58-64%	C
52-58%	C-
46-52%	D+
40-46%	D
34-40%	D-
below 34%	F



Academic Integrity Policy

- The USC Student Conduct Code *prohibits plagiarism*
  - for programming assignments, you must not *look at/copy/share* code fragments from *other students* or from *previous semesters*
- Lots of C and OS code on the Internet
  - you may use them if they are *general purpose* code (i.e., *not* written for CS 350 programming assignments)
  - if you use AI to generate code for our assignments, it's considered cheating since the generated code is not general purpose code

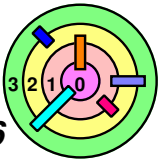


Academic Integrity Policy

- What if you end up with the same code fragment as another student because you copy the code fragment from the same place online (assuming that the code is not AI generated)?
  - it's a good habit to cite code you did not write yourself to give credit to your source
    - must cite the code *inline*

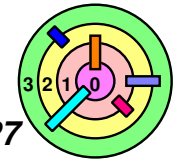
```
/* begin code from [URL] */
[ code you copied from above URL ]
/* end code from [URL] */
```

    - no good to give citation in your README file



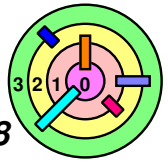
Academic Integrity Policy

- You can use any code *given to you* as part of *this class*
  - no need to cite such code
- You are encouraged to work with other students
  - "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 more details, please see the *Academic Integrity Policy* section on the *Course Description* web page



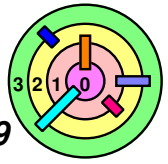
Displaying Your Code in Public Repositories

- 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 other students to verify
  - `github.com` is considered a *public repository* (no matter what they claim)
    - if you don't pay, your code becomes public
    - therefore, you *must not use* `github.com`
    - `bitbucket.org` is free for students and you can setup private repositories
- Since our programming assignments will be reused, you must not knowingly make your code public (not even pseudo-code)
  - *USC Student Conduct Code* says that you must not *cheat from other students* or *knowing help other students to cheat*
    - you have agreed to the USC Student Conduct Code
  - if I see your code posted in a public place, it would be considered a violation of USC Student Conduct Code



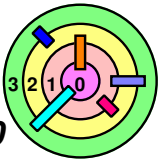
Displaying Your Code in Public Repositories

- 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*
    - it says so at the beginning of every assignment spec
  - the code given to you in our assignments are also private
    - they are private because access is password protected
  - *you must not post any of your CS 350 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*
  - this is serious business!
    - your future boss would/should appreciate that you understand about software copyrights



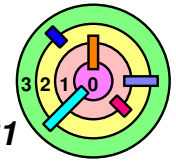
Program Checker

- Do not submit someone else's code either in whole or in part
- 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* the submissions from previous semesters
- If MOSS reports an unacceptable level of code match
  - I will 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



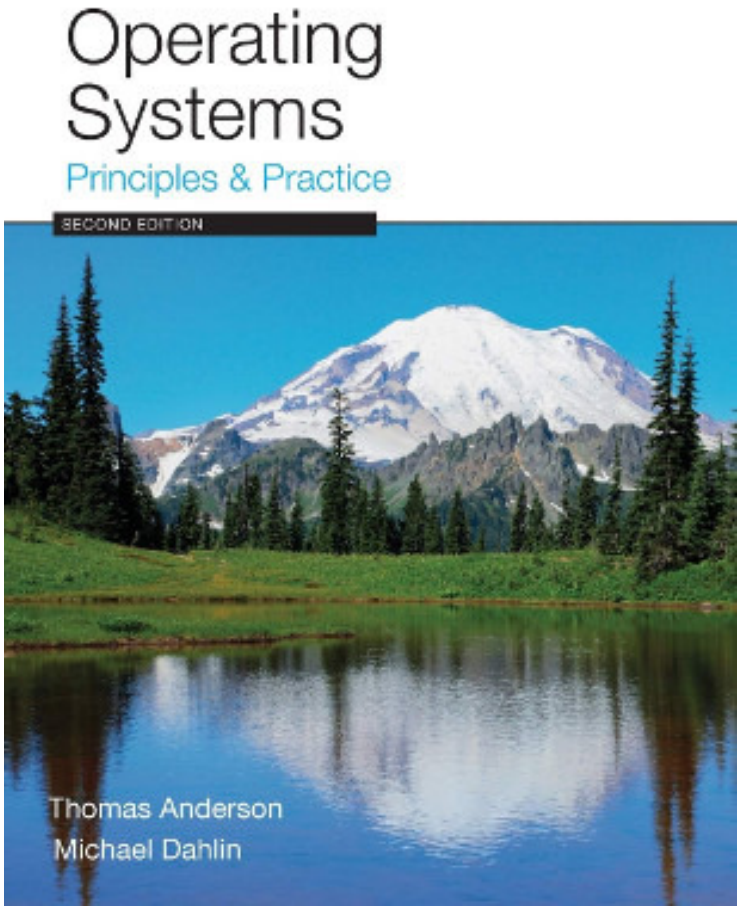
Email Questions

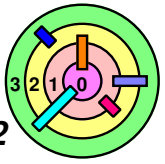
- One type of question I often get about assignments:
  - "I am thinking about not following the spec and do this instead. Is it acceptable (or is this okay)?"
    - my answer will always be: "stick to the written words (spec and grading guidelines)"
      - of course, if there is a conflict between the spec and the grading guidelines, you need to let me know and ask me to fix one of them



Course Readings

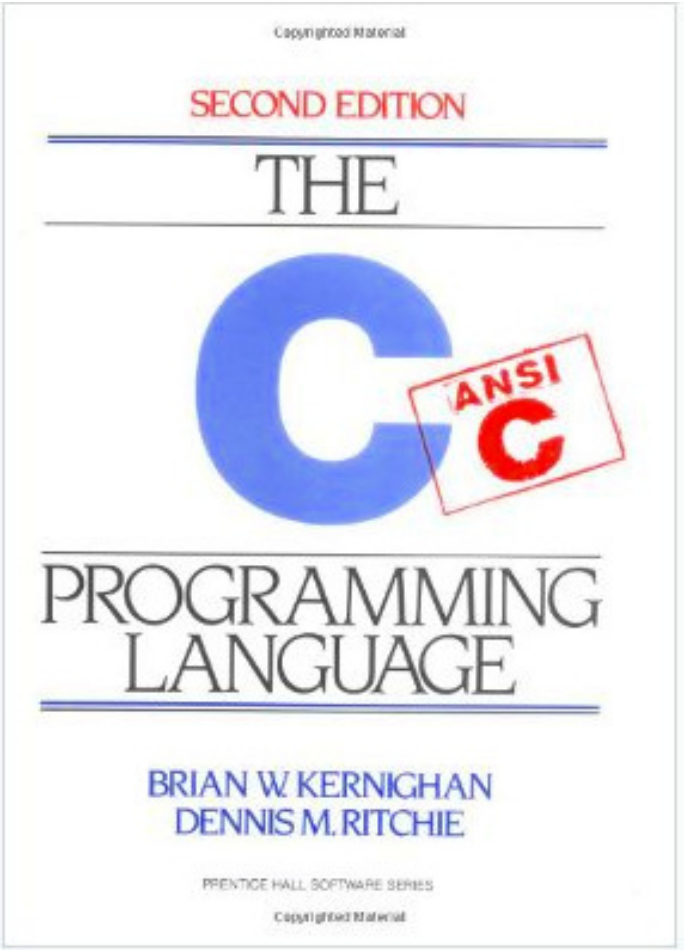
- ***Required*** **textbook**
  - ***"Operating Systems: Principles & Practice"*** **(2nd Edition) by T. Anderson & M. Dahlin**
  - **my plan is to mostly follow this book**



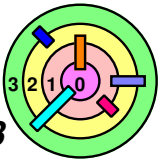


Course Readings

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

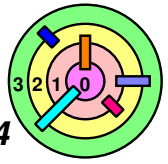


- Optional free online books
  - *"Pro Git"* by S. Chacon
  - *"Unix for the Beginning Mage"* by J. Topjian



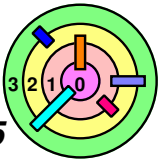
Class Structure

- Lectures mostly based on Anderson & Dahlin
  - lecture slides posted on class web site before class
  - please feel free to ask questions about them
- I expect you to attend every class meeting
  - if you do happen to miss a class, you are responsible for finding out what material was covered and what administrative announcements were made
  - you will be expected to take exams at scheduled times (although most likely, the exams will be take-home exams, to be completed during the pre-specified exam times)
    - you need to make sure that you are ***available*** to take exams at the scheduled times
      - no make-up exams
      - if you miss the midterm for any reason, your final exam will account for 70% of your overall grade



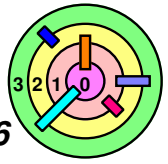
"Homeworks Assignments"

- There are "homework problems" at the end of each book chapter
  - we will not do them



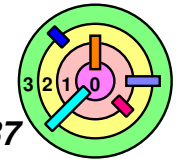
Programming Assignments

- 5 programming assignments *(must be done individually)*
  - (8%) PA1: add system calls to xv6
  - (23%) PA2: kernel level threads
  - (23%) PA3: mutex and condition variable
  - (23%) PA4: MFQ scheduler
  - (23%) PA5: memory management and copy-on-write
  - one assignment due every 2 weeks
- *No solutions will be given for any programming assignments*
  - PA3 depends on PA2
  - all other PAs start with the same xv6 base code
  - program in C only



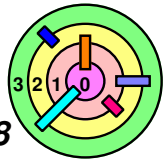
Programming Assignments

- All assignments can *only* be graded on a *"standard" 32-bit Ubuntu 16.04 system*
  - follow the instruction at the bottom of the class web page
  - two choices
    - install a virtual machine hypervisor, then install my *virtual appliance* into the hypervisor to create a virtual machine
      - VirtualBox if you have Intel/AMD CPU
      - UTM if you have Apple CPU (M1/M2/M3)
    - create an VM instance from *my AMI* on AWS Free Tier (free for one year if you don't go over the usage limit)
  - if you are offered to "upgrade" to a new release, *you must refuse*
    - or you have to reinstall another "standard" system
- You can do your development on another system
  - but you won't get any credit if your assignment cannot run on the "standard" system
    - no credit for just "effort"



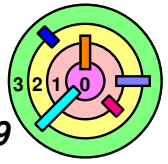
Programming Assignments

- One great thing about virtual machine is that you can install as many virtual machines as you'd like
  - keep one *clean* "standard" system for *testing* so you know exactly what the grader will see
- The "standard" system contains all the *pre-approved packages* you need to do all the PAs
  - once you install something else into this system, you no long have a "standard" system (i.e., it will be different from the one the grader will use)
  - if you ask the grader to install a package, the grader must refuse to do so
- It's a *really bad idea* to do a major upgrade of your host machine in the middle of the semester because that can break the system running inside a virtual machine
  - super important to make *daily backups*



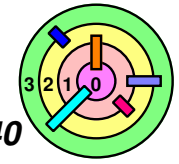
Programming Assignments

- In the README file you are required to submit, you must tell the grader on which system to grade
  1) standard 32-bit Ubuntu 16.04 (inside a virtual machine)
  2) AWS (which also runs a standard 32-bit Ubuntu 16.04)
  - these two systems are considered to be equivalent
    - if you notice any significant difference, please inform the instructor



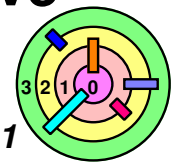
Programming Assignments

- The grader ***must follow*** the grading guidelines when grading programming assignments
  - due to our fairness policy, there is no other way
  - if you ask the grader to grade differently (using a different procedure or grade on a different system), the grader must refuse
    - ***the grader has only one way to grade and that's by following the grading guidelines***
      - do expect double-jeopardy, triple-jeopardy, etc.
      - the grader is not permitted to give credit based on "effort"
    - it's your responsibility to make sure that your code passes all the tests in the grading guidelines on the "standard" system
  - if you are not sure about something, please ask me and don't just assume that whatever you do will be acceptable
- It's super important to start your PAs ***early***
  - don't under-estimate the amount of time you need to spend to get your code to work perfectly



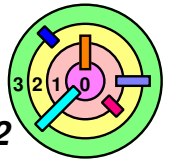
Programming Assignments Grading

- All input data files for programming assignments will be correctly formatted
  - i.e., you can assume that they are perfect
  - do you have to validate input data?
    - you don't have to, *IF* your code is perfect
    - but if you don't validate input, what if your code has bugs?
      - you may end up with a very low score
    - it's best you validate all input data
      - if your code thinks that an input file is bad, it's your responsibility to inform the instructor as soon as possible
      - if the input is bad, I will fix it and make an announcement
      - if the input is good, you must fix your bug
- In previous semesters, some students' code would abort on all input (and pass no tests) and still want partial credit for "effort"
  - the grader must follow grading guidelines and we cannot give partial credit just based on "efforts"
    - you can only get partial credit for ***passing tests***



Electronic Submissions

- Use the *Bistro* system (see bottom of PA 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 system timestamp*
    - we can only trust things that have made it to my 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 each binary file submitted)
  - it is *your responsibility* to make sure that your submission is what you want us to grade - *Be a little paranoid!*

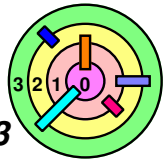


Electronic Submissions

- Use the *Bistro* system (see bottom of lab and PA 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 (timestamp of an email submission is *my receiving email server's timestamp* and not your sending email server's timestamp)
- *Verify* what you have submitted
  - if your submission file is "`pa1.tar.gz`", the `hash_value` in the *ticket* should match the printout of the following command:

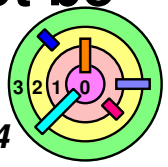
```
openssl sha1 pa1.tar.gz
```

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

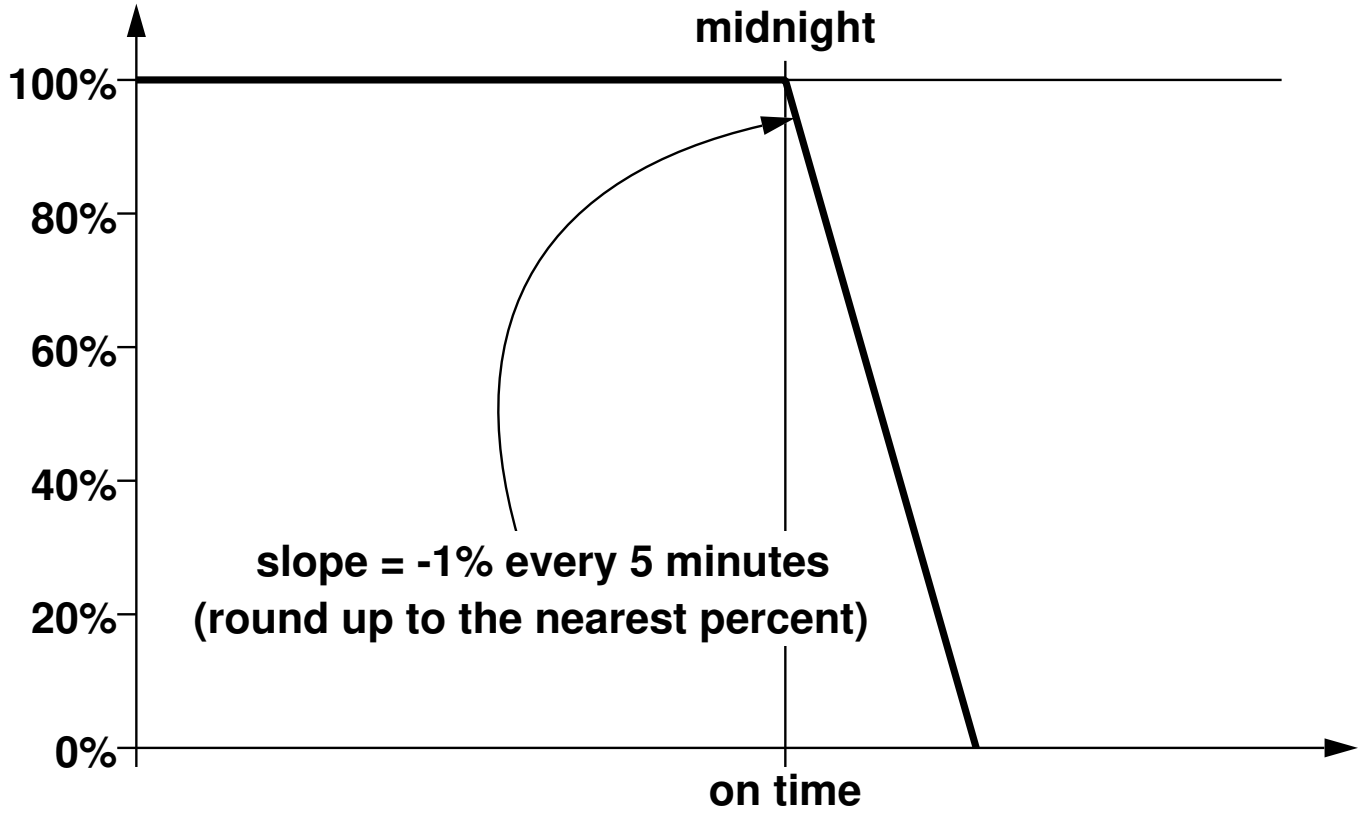


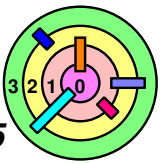
Late Policy For Programming Assignments

- Programming assignments submitted electronically
  - 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



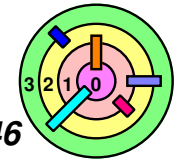
Late Policy For Programming Assignments





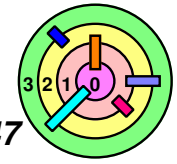
Late Policy For Programming Assignments

- Each student has 5 "free late days" that can be applied to any PAs
  - you decide how you want to distribute these 5 "free late days"
    - you can even put them all on one PA
    - for PA5, you can use at most 1 "free late day" because we need to get grading started as early as possible
  - if you have used a "free late day" on an assignment, *once that assignment is graded*, you *cannot "un-use"* that "free late day"
- A "free late day" can only be used on a *late submission*



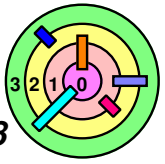
Late Policy For Programming Assignments

- For a PA, you can submit multiple times, only the last submission will be graded (unless you send the instructor an email to ask us to grade an earlier submission)
  - please do not send such a request to a TA
- For PA1, 2, 3 or 4, you can submit before the next PA submission deadline for a 50% deduction
  - requirement: you must inform the instructor before the PA grade is sent to you
    - once you have received a grade for a PA, this option is not available for that PA
      - therefore, if you have made a submission already, you should send email to the instructor to withdraw that submission (i.e., tell us not to grade that submission)



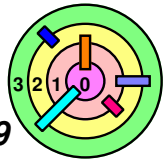
Late Policy For Programming Assignments

- 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



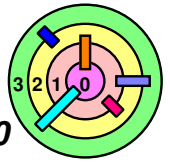
Modifications After Deadline for PAs

- 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 assignment 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 assignment submission deadline
  - (cont...)



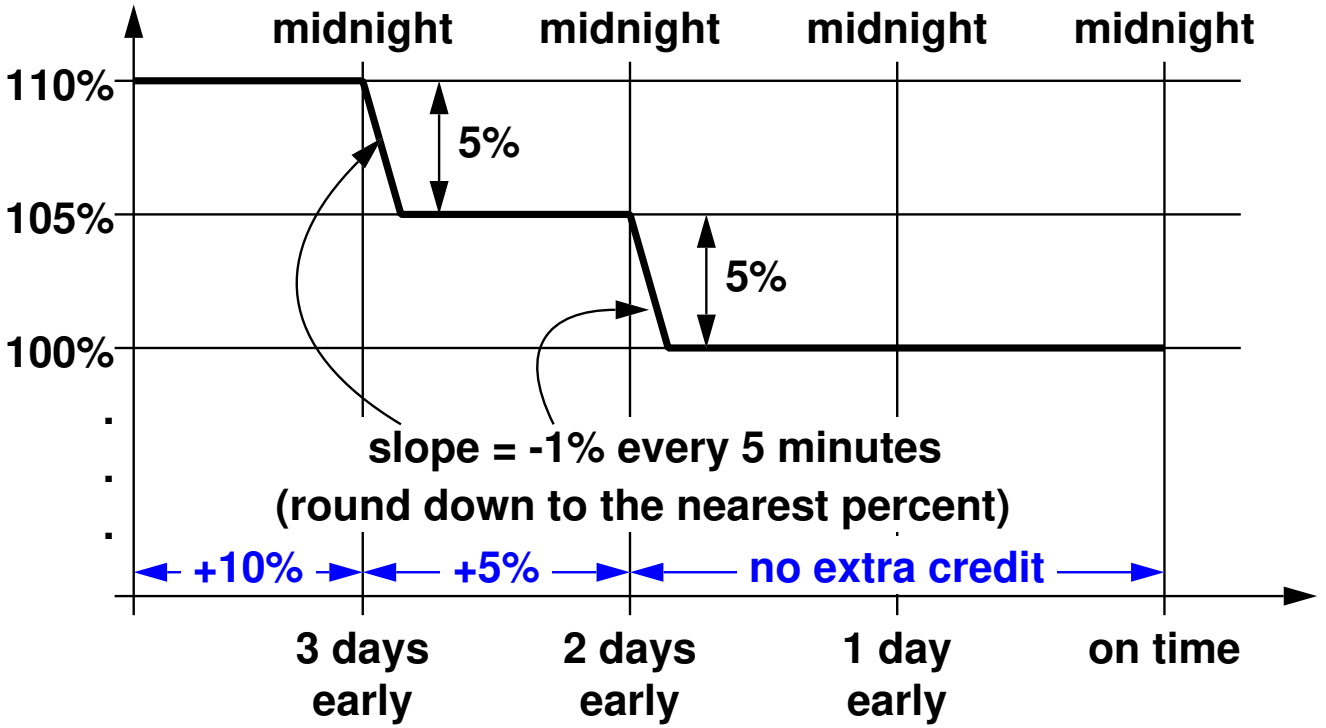
Modifications After Deadline for PAs

- After the submission deadline has past (cont...)
  - 24 hours after the submission deadline, additional modifications cost *6 points per line* for the next 6 days
  - afterwards, it costs *15 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 (by following the Verify Your Submission procedure) after you have made your final submitted to be sure



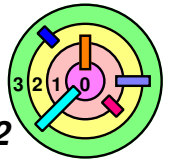
Extra Credit

- To encourage finishing programming assignments early, you will get extra credit if you turn in a *PA* more than 2 or more 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%*



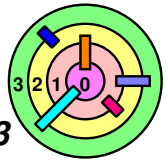
Regrade Policy

- 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 email
  - 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



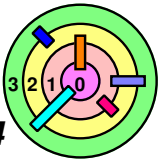
Student Commitments

- Keep up with your course work
- Do your own work
- Turn in lab assignments on time and start PAs *early*
- Turn in PAs on time (preferably before extra credit deadline)
  - you are encouraged to ask me questions, pretty much about anything related to programming
- 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
  - *verify* your ticket and *verify* your submission, 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 code or pseudo-code) about PAs



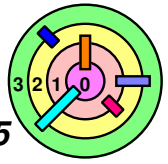
Student Commitments

- If you feel that you are falling behind
  - talk to the instructor as soon as possible
- You need to manage your time well and start all your assignments as early as you can
  - some students only start a programming assignment a couple of days before the submission deadline
    - if you do that, it's very likely that you will get a very very low score
      - the assignments may be harder than they look
      - the grader must follow grading guidelines when grading



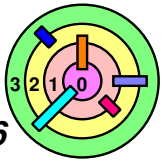
Exams

- Exams are worth a lot
  - this is *not* just a programming class, even though you may end up spending a lot of time programming
- Exams will be mostly based on lectures (and corresponding material in the textbook)
  - I may also ask you about lab and programming assignment (higher level concepts, won't ask you to code)
  - I reserve the right to ask anything based on required materials that I think you should know
  - to do well, you have to *study* to understand the materials well and be able to *think clearly* about the concepts learned
- If you want to get a good grade in this class, you have to do well in the exams and assignments
  - just do well in the programming assignments is not enough



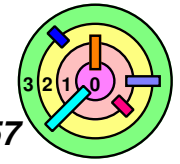
Too Much Time Programming

- Some students complained that they had to spend too much time programming and debugging
  - that's really not supposed to happen
    - our assignments are "thinking-type" assignments
    - always *write your best code with full intent* and don't take the approach of trying to hack your code until it works because that approach doesn't work well in general anyway
    - you need to *understand* the spec *before you start coding* and know how things are supposed to work
      - if the specs are not clear, ask me questions before you write a bunch of code



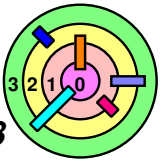
Too Much Time Programming

- Some students complained that they had to spend too much time programming and debugging (cont...)
  - 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 this line of code (or function call) *fail* and did I handle all the failed cases correctly?
      - this is *extremely important in OS code*
    - 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 never out of bounds?
  - it may be a good idea to write code incrementally (and don't write everything in one shot and expect everything to just work because that may not be realistic)



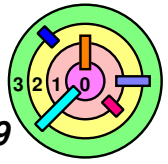
Too Much Time Programming

- 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* (although sometimes error messages are difficult to read)
    - much better to *avoid creating bugs* than to *debug*
      - this can *save you a lot of time*



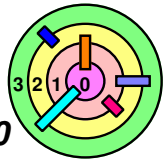
Things to Do Today

- Read entire class web page: http://merlot.usc.edu/cs350-m25
  - 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 PA1 specs and *start coding*
- Additional things to learn/do quickly
  - learn how to use a command shell in a "terminal"
  - learn a commandline editor: `vim/pico/emacs`
    - `pico` is the easiest
  - install a *standard 32-bit Ubuntu 16.04* system into VirtualBox/UTM or AWS Free Tier
    - ask for help if there are problems
    - the AWS Free Tier system is actually a pretty usable system for this class
      - it's good to have that on your resume



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 SSO authentication* is *required* for *all Zoom meetings*
  - 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
      - log out of Zoom then log back in using USC SSO authentication

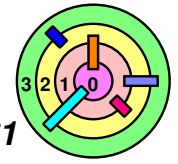


Course Content Credit

- Slides and course content primarily came with the textbook and originally written by T. Anderson & M. Dahlin
- Additional slides and course content may have come from:
  - Tatyana Ryutov

Miscellaneous

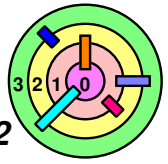
- Someone actually complained about this, so I'm stating it here
  - I do not use the standard course evaluation window
    - in the old days, course evaluations are done in class (i.e., the window was about 15 minutes)



CS 350
PA1: Add System Calls To XV6

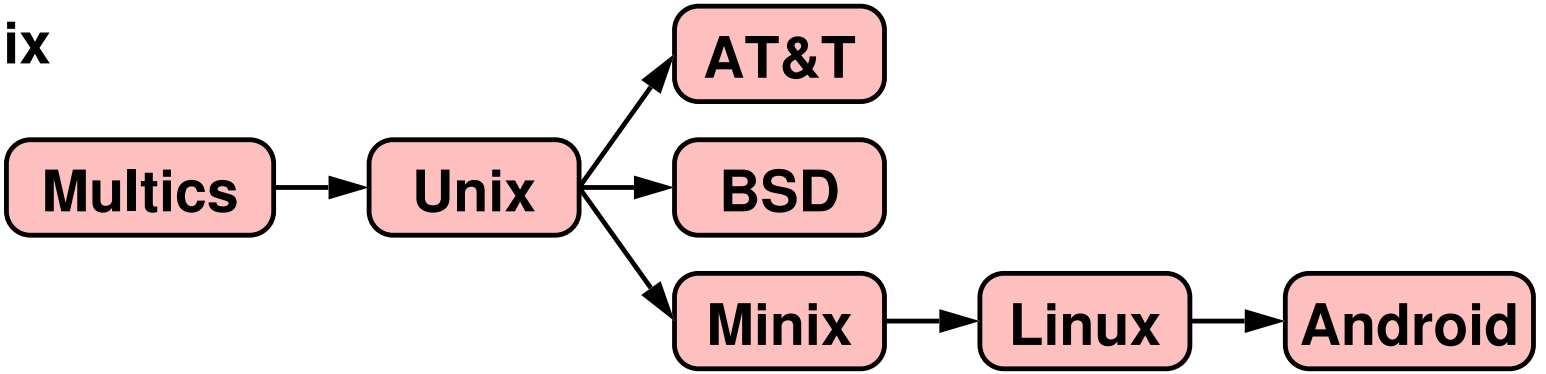
Bill Cheng

http://merlot.usc.edu/william/usc/

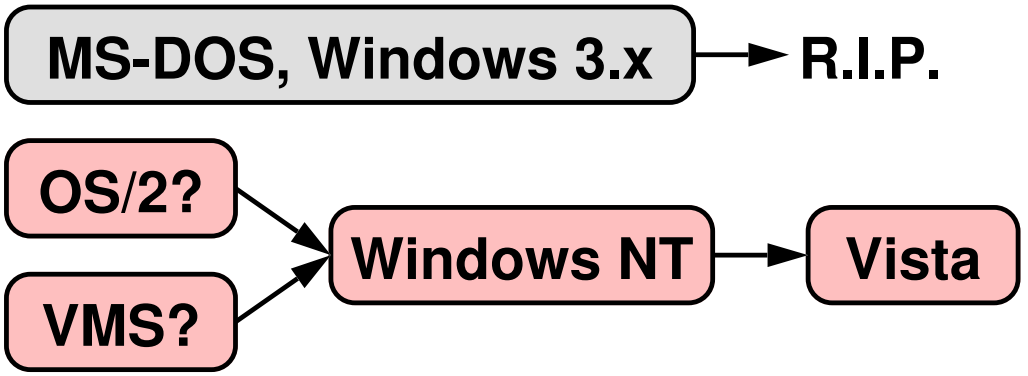


Is Unix/Linux Still Relevant Today?

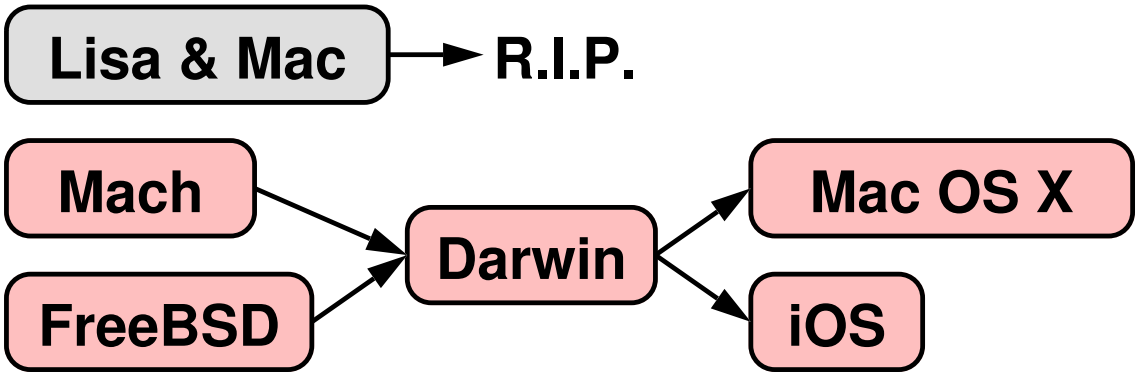
- Linux & Unix

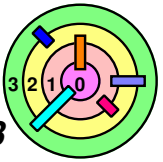


- Microsoft



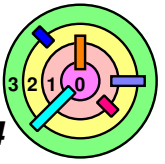
- Apple





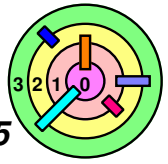
What Is C?

- Unix/Linux *system calls* have a C functional interface
  - must use a system call to use hardware
- Early 1960s: CPL (Combined Programming Language)
  - developed at Cambridge University and University of London
- 1966: BCPL (Basic CPL): simplified CPL
  - intended for systems programming
- 1969: B: simplified BCPL (stripped down so its compiler would run on minicomputer)
  - used to implement earliest Unix
- Early 1970s: C: expanded from B
  - motivation: they wanted to play "Space Travel" on minicomputer
  - used to implement all subsequent Unix OSes
- Unix has been written in C ever since



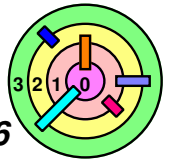
PA1

- Set up a standard 32-bit Ubuntu 16.04 system
  - download xv6 and get familiar with xv6
- Part 1 - add a new system call
  - `trace()`
- Part 2 - add a second system call
  - `date()`



Some Basic Linux Commands

```
ls
ls -a
ls -l
echo "hello"
echo -n "hello"
echo `date`
echo `date +%m%d%y-%H%M%S`
cat /etc/os-release
more /etc/os-release
mkdir tmp
pwd
cd tmp
pwd
ls
cd ..
cp /etc/os-release tmp
cp tmp/os-release tmp/abc
ls -aF tmp
mv tmp/os-release tmp/xyz
ls -aF tmp
diff tmp/abc tmp/xyz
man gcc
rm tmp/abc
touch tmp/defg
ls -alF tmp
ps -x
ps -auxw
pico tmp/xyz
rm tmp/defg tmp/xyz
ls -alF tmp
rmdir tmp
ls -alF tmp
df
top
exit
```

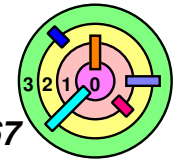


Notes On gdb

- The debugger is your friend! Get to know it *NOW*!

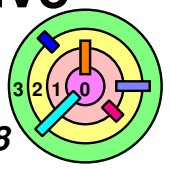
start debugging:	`gdb`
list source code:	`(gdb) list`
set breakpoint:	`(gdb) break foo.c:123`
list all breakpoints:	`(gdb) info breakpoints`
continue:	`(gdb) cont`
clear breakpoint:	`(gdb) clear`
stack trace:	`(gdb) where`
print field:	`(gdb) print f.BlockType`
print in hex:	`(gdb) print/x f.BlockType`
single-step at same level:	`(gdb) next`
single-step into a function:	`(gdb) step`
print field after every cmd:	`(gdb) display f.BlockType`
assignment:	`(gdb) set f.BlockType=0`
quit:	`(gdb) quit`

- Start using the debugger with PA1!
  - get help from TAs and me



Set Up A Standard System

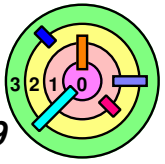
- Go to class home page, scroll all the way to the bottom and click on the *install a standard 32-bit Ubuntu 16.04 system* link
- You have two choices
  1) install a virtual machine hypervisor, then install my *virtual appliance* into the VM hypervisor to create a virtual machine
     - install *VirtualBox* if you have Intel/AMD CPU
     - install *UTM* if you have Apple CPU (M1/M2/M3)
  2) create an VM instance from *my AMI* on *AWS Free Tier* (free for one year if you don't go over the usage limit)
  - if you like to do development on your host machine, you need to figure out a way to transfer files between your host machine and the "standard" system
    - my recommendation is to use FileZilla
    - some would like to use SSH in VScode
      - this would only work with (1) above and you need to give 4 GB RAM becuase VScode is memory hungry



Download XV6

- Follow the instructions on the PA1 spec
- Open a *terminal* and type the following

```
cat /etc/os-release
pwd
mkdir cs350
cd cs350
mkdir pa1
cd pa1
wget --user=USERNAME --password=PASSWORD \
  http://merlot.usc.edu/cs350-m25/programming/pa1/xv6-pa1-src.tar.gz
tar xvf xv6-pa1-src.tar.gz
cd xv6-pa1-src
ls
```



Run XV6

- Three ways to run xv6:
  1) run xv6 console in a separate window:

```
make qemu
```

  - I do not recommend this way

  2) run xv6 in commandline mode:

```
make qemu-nox
$ ls
$ echo Hello
$ cat README
```

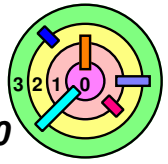
  3) debug xv6 commandline mode:

```
make qemu-nox-gdb
```

  - in a separate terminal, do:

```
gdb
(gdb) source .gdbinit
(gdb) list exec.c:11
(gdb) break exec
(gdb) break sys_open
(gdb) cont
```

  - by default, you are debugging the kernel



Debugging User Space Code In XV6

- The last line in .gdbinit says to debug the kernel

```
(gdb) symbol-file kernel
```

  - sometimes, you may need the assembly listing of the kernel

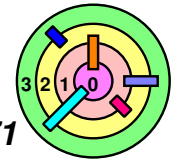
```
objdump --disassemble --section=".text" kernel > kernel.txt
objdump --disassemble --section=".text" -S kernel > kernel.txt
```

    - you can open `kernel.txt` with your favorite text editor
  - switch to debug the "`ls`" user space program

```
(gdb) symbol-file _ls
(gdb) list 25
(gdb) break ls
(gdb) break open
(gdb) cont
```

  - in the first window, when you get the xv6 prompt, type "`ls`" to run the "`ls`" program
    - you should break at the beginning of the `ls()` function

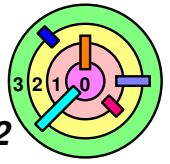
```
(gdb) c
Continuing.
```



Debugging User Space Code In XV6

- you should break at the `open()` functionsystem call
  - since `open()` is a ***system call***, things would look different

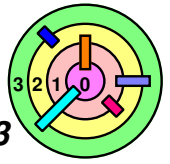
```
[  1b: 5c2]    0x772 <printf+162>:      in     (%dx),%al

Thread 1 hit Breakpoint 2, open () at usys.S:20
20      SYSCALL(open)
(gdb) list usys.S:20
15      SYSCALL(read)
16      SYSCALL(write)
17      SYSCALL(close)
18      SYSCALL(kill)
19      SYSCALL(exec)
20      SYSCALL(open)
...
(gdb) list usys.S:9
4       #define SYSCALL(name) \
5         .globl name; \
6         name: \
7           movl $SYS_ ## name, %eax; \
8           int $T_SYSCALL; \
9           ret
10
11      SYSCALL(fork)
...
```



Debugging User Space Code In XV6

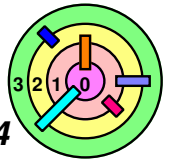
- An application program doesn't know how to open a file
  - why now?
  - only the OS kernel knows how to do that
  - to ask the OS kernel for help, you make a system call
  - in xv6, the convention is that if `foo()` is a *system call*, the corresponding OS kernel function is called `sys_foo()`
- To go from user space code to the kernel requires a *context switch*
  - there are different types of context switches
  - here we switch from the *user space context* to the *kernel space context*
    - we will talk more about this in class
    - for now, you need to switch to debug kernel code

```
(gdb) delete
(gdb) symbol-file kernel
(gdb) break sys_open
(gdb) cont
```



Debugging User Space Code In XV6

- The kernel versions of the system calls that are related to the file system is in `sysfile.c`
  - `sys_open()` looks like regular C code
    - but remember, you are now in the all power kernel, you can really mess things up if you are not careful
- Gdb is designed to mainly debug regular C code
  - it's not comfortable with switching contexts
    - single-step gdb command (i.e., "`next`" and "`step`") may not work as expected when a context switch is involved
    - if you know that a context switch will happen, you should set a breakpoint and use the "`cont`" gdb command to get there
  - although if you switch to assembly level debugging, then gdb will just be debugging machine code and it won't worry about context switching
    - why not?
    - context switching is just an ***abstraction***



Debugging User Space Code In XV6

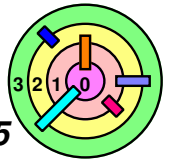
- How to get back to user space code?
  - you need to get back to where you made the `open()` system call
  - open `ls.c` and see that you called `open()` on line 33

```
(gdb) delete
(gdb) symbol-file _ls
(gdb) list ls.c:34
29          int fd;
30          struct dirent de;
31          struct stat st;
32
33          if((fd = open(path, 0)) < 0){
34            printf(2, "ls: cannot open %s\n", path);
35            return;
36          }
37
38          if(fstat(fd, &st) < 0){
```

  - therefore, you should do:

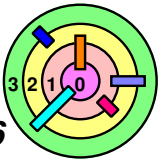
```
(gdb) break ls.c:34
(gdb) break ls.c:38
(gdb) cont
```

  - now you are back in user space



Advanced Debugging In XV6 (Optional)

- ***if you really want to know*** **how context switching works from user space to kernel space, you need to switch to debug assembly code (you probably have seen this in CS 356)**
  - ***"Abandon all hope, ye who enter here".***



Advanced Debugging In XV6 (Optional)

- ***if you really want to know*** **how context switching works from user space to kernel space, you need to switch to debug assembly code (you probably have seen this in CS 356)**
  - ***"Abandon all hope, ye who enter here".***

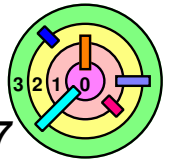
```
(gdb) layout asm
```

- **the window splits and the top panel would look like:**

```
   +-------------------------------------------
B+>|0x5c2 <open>              mov    $0xf,%ax
   |0x5c5 <open+3>            add    %al,(%bx,%si)
   |0x5c7 <open+5>            int    $0x40
   |0x5c9 <open+7>            ret
   |0x5ca <mknod>             mov    $0x11,%ax
   |0x5cd <mknod+3>           add    %al,(%bx,%si)
   |0x5cf <mknod+5>           int    $0x40
   |0x5d1 <mknod+7>           ret
   +-------------------------------------------
```

- **single step at the assembly code level:**

```
(gdb) si
(gdb) si
```



Advanced Debugging In XV6 (Optional)

- the top panel now looks like:

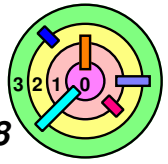
```
+-------------------------------------------
>|0x80105cf9      push   $0x40
 |0x80105cfb      jmp    0x8010560a
 |0x80105d00      push   $0x0
 |0x80105cf9      push   $0x41
 |0x80105cfb      jmp    0x8010560a
 |0x80105d00      push   $0x0
 |0x80105cf9      push   $0x42
 |0x80105cfb      jmp    0x8010560a
 |0x80105d00      push   $0x0
+-------------------------------------------
```

- they correspond to the following in "`usys.S`":

```
20      SYSCALL(open)
21      SYSCALL(mknod)
22      SYSCALL(unlink)
```

- set a breakpoint at virtual address `0x8010560a`

```
(gdb) break *0x8010560a
(gdb) cont
```



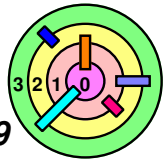
Advanced Debugging In XV6 (Optional)

- what's at `0x8010560a`?
- open `kernel.txt` and do a string search for `8010560a`

```
8010560a <alltraps>:
8010560a:       1e                      push   %ds
8010560b:       06                      push   %es
8010560c:       0f a0                   push   %fs
8010560e:       0f a8                   push   %gs
80105610:       60                      pusha
80105611:       66 b8 10 00             mov    $0x10,%ax
80105615:       8e d8                   mov    %eax,%ds
80105617:       8e c0                   mov    %eax,%es
80105619:       54                      push   %esp
8010561a:       e8 e1 00 00 00          call   80105700 <trap>
```

- in `kernel.txt`, `<trap>` looks code generated by a C compiler

```
80105700 <trap>:
80105700:       55                      push   %ebp
80105701:       89 e5                   mov    %esp,%ebp
...
```



Advanced Debugging In XV6 (Optional)

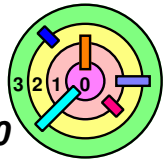
- to set a breakpoint there, you need to clear all breakpoints, switch to debug the kernel, and set a breakpoint there

```
(gdb) delete
(gdb) symbol-file kernel
(gdb) break trap
(gdb) layout src
(gdb) cont
```

- to get rid of the top panel:

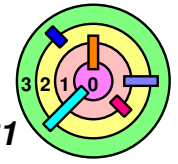
```
(gdb) tui disable
```

⇨ The assembly code level debugging is optional for now
  - hopefully, you won't need to do that in this class
  - one day, when you have a really tough bug and the only way to debug your code is to debug context switching at the assembly code level, then you need to come back here and review all this



Part 1: Add A System Call

- You need to read the xv6 book in the spec to understand how xv6 works
  - Ch 3 contains details on traps and system calls (although most of the low level details *are not needed* for you to complete this assignment)
- Your job is to add a new system call called `trace()`
  - since you know the basic flow from `open()` to `sys_open()` and back to `open()`, you should be able to add a `trace()` system call to reach `sys_trace()` and get back to `trace()`
    - of course, you need to implement `sys_trace()` according to the spec



Part 1: Add A System Call

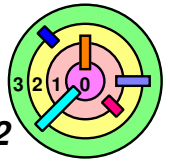
- Which files do you need to modify?
  - open a terminal and type the following:

```
pwd
cd cs350/pa1/xv6-pa1-src
make -n pa1-submit
```

    - you should see:

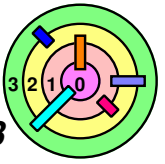
```
tar cvzf pa1-submit.tar.gz \
 Makefile \
 pa1-README.txt \
 proc.c \
 proc.h \
 syscall.c \
 syscall.h \
 sysproc.c \
 user.h \
 usys.S
```

  - these are the only files are are supposed to submit
    - if you submit additional files, the grader will have to delete them before grading
    - if you submit binary files, points will be deducted



Part 1: Add A System Call

- `test_project1.c` is a test program for part 1
  - need to modify `Makefile` get it compiled so the grader can run it
- Please take a look at the grading guidelines to see what the grader will do to grade part 1
  - `grade_pa1.c` is another test program for part 1
    - need to include that in your `Makefile`



Part 2: Add Another System Call

- Your job is to add a new system call called `date()`
  - need to call `cmostime()` to get read the real time clock (which is the current UTC time
- `date.c` is a test program for part 2
  - need to modify `Makefile` get it compiled so the grader can run it