

CS162, Spring 2002  
Section 1  
Steve Martin

### Introductions

- Who I am
- What section will typically be like
- Section homepage, old section homepages, etc

### Class logistics

- SECTION IS TEMPORARY: Will be assigned to new section based on
- project group
- Project groups: 4-5 people in each group
  - o ALL MEMBERS MUST BE ABLE TO ATTEND SAME SECTION
- Sign up for project groups immediately on the web page
- Need your cs162 class accounts to do this.
  - o Log in as soon as possible.

### Newsgroup

- ucb.class.cs162
- Please use the newsgroup rather than e-mailing me

### Office Hours

- Tuesday 10-11am, Friday 11am-12pm, ??? Soda (Location TBD)

### Exams

- 1 midterm, 1 final.
  - o See course web for details.

### Book

- Silberschatz and Galvin, Operating Systems Concepts, 6th Edition (see web page)
- Reader with the Nachos code so you don't have to print it out
  - o Copy Central at Northside (probably)

### Projects

- Get Nachos from Homepage
  - o Unpack and run it. Browse the source and trace the control flow!
- 1-1.5 weeks for design
  - o 15-20 minute design meeting with group - everyone in group MUST attend
    - Sign up sheet for meetings will be online and will be posted
- 1-1.5 weeks for implementation
  - o START EARLY! May have problems with getting Nachos, CVS, etc. working, especially first time
- Deadline will be 11:59pm on the day the assignment is due – use 'submit' program
- You get a total of 5 slip days during the semester
  - o Rounded up to next full day (so 12:00 midnight counts as 1 day)

- No excuses for late work after you use your slip days!
- 'glookup' program to look up your grades

### Nachos

- Used for our projects, by ex-Berkeley prof
- Easy to use, makes OS development experimentation much easier than, say, hacking Linux
- Implemented in Java
- Runs on UNIX, Windows, etc - very portable
- Nachos roadmap online

### CVS

- Groups should use CVS for sharing code during projects
- Great way to manage the project (and also serves to back up everything you do!)
- See the web page for a link to a tutorial

### Lecture Review...

*What is an OS?*

- Coordinator, facilitator
- Abstractions, VMs
- Address Translation
- Dual mode operation

*Why can an OS be considered a 'coordinator' of services? What kinds of services does the OS control?*

An OS mediates programs' access to hardware resources

- Computation (CPU)
- Volatile storage (memory) and persistent storage (disk, etc.)
- Network communications (TCP/IP stacks, ethernet cards, etc.)
- Input/output devices (keyboard, display, sound card, etc.)

*What does the OS abstract for users?*

The OS abstracts hardware into logical resources and well-defined interfaces to those resources

- processes (CPU, memory)
- files (disk)
- programs (sequences of instructions)
- sockets (network)

*What is Dual Mode Operation? Why do we need this?*

To support multiprogramming! (would we need this otherwise?)

*Why bother with an OS?*

- programming simplicity
- portability (across machine configurations or architectures)
- safety
  - o program “sees” own virtual machine, thinks it owns computer
  - o OS protects programs from each other
  - o OS fairly multiplexes resources across programs
- efficiency (cost and speed)
  - o share one computer across many users
  - o concurrent execution of multiple programs

*Major OS Issues*

- **structure**: how is the OS organized?
- **sharing**: how are resources shared across users?
- **naming**: how are resources named (by users or programs)?
- **security**: how is the integrity of the OS and its resources ensured?
  - o **protection**: how is one user/program protected from another?
- **performance**: how do we make it all go fast?
- **reliability**: what happens if something goes wrong (either with hardware or with a program)?
- **extensibility**: can we add new features?
- **communication**: how do programs exchange information, including across a network?
- **concurrency**: how are parallel activities (computation and I/O) created and controlled?
- **scale**: what happens as demands or resources increase?
- **persistence**: how do you make data last longer than program executions?
- **distribution**: how do multiple computers interact with each other?
- **accounting**: how do we keep track of resource usage, and perhaps charge for it?

*Semi-Bonus Question: Should a user program be allowed to change the system clock? Why or why not?*