Lecture 1:
Introduction & OS Structures
(Part One, Chapters 1&2)
Class Resources

- Class web page: http://www.cse.usf.edu/~anda/cop6611
  - Syllabus
  - Research papers
  - Schedule
  - Announcements

- Textbook
  - Complete chapter slides (maybe good reviewing notes) at: http://os-book.com

- Work submission:
  - Mostly Blackboard
  - Reading summaries on H2O
Calibration Test

Next classes:
- Thursday: reading due on Exokernel paper
- Tuesday: reading due on Microkernel paper (+ Ch 1,2 textbook)
- Following class (Th): reading due on Xen paper

Bring papers to class for discussions

Action items for you:
- Join H2O project (http://h2o.law.harvard.edu/ViewProject.do?projectId=20687)
  - No fancy formatting, no attachments, keep it simple
  - Forum restricted to class students (will enforce before next class)
- Submit reading in Exokernel paper before Th at 10am
  - Regular submission times for reading is midnight before Th class
Part One: Overview (selected topics)

- What Operating Systems Do
- Process Management
- Memory Management
- Storage Management
- Protection and Security
- Operating System Services
- User Operating System Interface
- System Calls, Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure

Terminology:
- Multiprogramming
- Spooling
- Paging, page vs. frame
- Address space
- Virtual memory
- Threads
- Scheduling
- Preemptive/non-~ threads
A View of Operating System Services

<table>
<thead>
<tr>
<th>user and other system programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI</td>
</tr>
<tr>
<td>user interfaces</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>system calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>program execution</td>
</tr>
<tr>
<td>error detection</td>
</tr>
</tbody>
</table>

operating system

hardware
User vs. Kernel Mode
Example of System Calls

- System call sequence to copy the contents of one file to another file

Example System Call Sequence
- Acquire input file name
- Write prompt to screen
- Accept input
- Acquire output file name
- Write prompt to screen
- Accept input
- Open the input file
  - if file doesn't exist, abort
- Create output file
  - if file exists, abort
- Loop
  - Read from input file
  - Write to output file
  - Until read fails
- Close output file
- Write completion message to screen
- Terminate normally
API – System Call – OS Relationship

user application

open ()

system call interface

open ()
Implementation of open ()
system call

return
Standard C Library Example

- C program invoking printf() library call, which calls write() system call

```c
#include <stdio.h>
int main()
{
    ...
    printf("Greetings");
    ...
    return 0;
}
```
Start by defining goals and specifications

- Affected by choice of hardware, type of system
- User goals and System goals
- User goals – convenient to use, easy to learn, reliable, safe, and fast
- System goals – easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

Important principle to separate

**Policy:** What will be done?
**Mechanism:** How to do it?

Mechanisms determine how to do something, policies decide what will be done

- The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later
Exokernel Paper

- OSes as we know them:
  - Diverse user goals
  - Diverse usage of resources
  - Possibly diverse hardware resources (e.g., disks)
  - Provide resource management and resource protection
  - ... by defining abstractions and limiting user control
- Exokernel paper says:
  - Abstractions are costly, therefore OS operations are unnecessarily costly
  - Solution:
    - Drastically reduce the kernel by separating resource management from protection
    - Export much of the traditional OS services from kernel space to user space
    - User space: deals with resource management
    - Kernel space: deals with resource protection