CMPS 111 - Operating Systems

Prof. Scott Brandt
Computer Science Department
University of California, Santa Cruz
General Information

• Focus is on operating systems
  – Complies with ACM & IEEE courses
  – Prerequisites: CMPS 101 & CMPW 12C

• Labs will have new material in them

• Do your work in the Unix Lab -- AS 105

• OK to discuss assignments, but:
  – Must develop your own code
  – Cannot look at other’s code
  – Cannot use code in a book
General Information (cont)

• Course grade
  – 4 programming assignments (40%)
  – 2 Quizzes (10%)
  – Midterm (20%)
  – Final (30%)

• Get all information from web page

http://www.cse.ucsc.edu/~sbrandt/courses/Spring00/111
Introduction
Why Study OS?

• Understand *model of operation*
  – Easier to see how to use the system
  – Enables you to write *efficient* code

• Learn to design an OS

• Even so, OS is pure overhead of real work

• Application programs have the real value to person who buys the computer
System Software

• Independent of applications, but common to all

• Examples
  – C library functions
  – A window system
  – A database management system
  – Resource management functions
Purpose of an OS
(What is Resource Management?)

• **Process**: An executing program

• **Resource**: Anything that is needed for a process to run
  – Memory
  – Space on a disk
  – The CPU

• “An OS creates resource abstractions”

• “An OS manages resource sharing”
Resource Abstraction

load(block, length, device);
seek(device, 236);
out(device, 9)
Resource Abstraction

```c
load(block, length, device);
seek(device, 236);
out(device, 9)
```

```c
write(char *block, int len, int device,
     int track, int sector) {
    ...
    load(block, length, device);
    seek(device, 236);
    out(device, 9);
    ...
}
```
Resource Abstraction

```c
load(block, length, device);
seek(device, 236);
out(device, 9)

write(char *block, int len, int device,
       int track, int sector) {

    ...
    load(block, length, device);
    seek(device, 236);
    out(device, 9);
    ...
}

write(char *block, int len, int device, int addr);
```
Resource Abstraction

load(block, length, device);
seek(device, 236);
out(device, 9)

write(char *block, int len, int device,
    int track, int sector) {
    ...
    load(block, length, device);
    seek(device, 236);
    out(device, 9);
    ...
}

write(char *block, int len, int device, int addr);

fprintf(fileID, "%d", datum);
Resource Sharing

- Space- vs time-multiplexed sharing
- To control sharing, must be able to isolate resources
- OS usually provides mechanism to isolate, then selectively allows sharing
  - How to isolate resources
  - How to be sure that sharing is acceptable
- Concurrency
Multiprogramming

- Technique for *sharing* the CPU among *runnable* processes
  - Process may be *blocked* on I/O
  - Process may be *blocked* waiting for other resource
- While one process is blocked, another should be able to run
- Multiprogramming OS accomplishes CPU sharing “automatically”
- Reduced time to run all processes
How Multiprogramming Works

Space-multiplexed Memory

Time-multiplexed CPU
OS Strategies

- Batch processing
- Timesharing
- Personal computer & workstations
- Process control & real-time
- Network
- Distributed
Batch Processing

- Uses multiprogramming
- *Job* (file of OS commands) prepared offline
- Batch of jobs given to OS at one time
- OS processes jobs one-after-the-other
- No human-computer interaction
- OS optimizes resource utilization
- Batch processing (as an option) still used today
Timesharing

- Uses multiprogramming
- Support interactive computing model (Illusion of multiple consoles)
- Different scheduling & memory allocation strategies than batch
- Tends to propagate processes
- Considerable attention to resource isolation (security & protection)
- Tend to optimize response time
Personal Computers

- CPU sharing among one person’s processes
- Power of computing for personal tasks
  - Graphics
  - Multimedia
- Trend toward very small OS
- OS focus on resource abstraction
- Rapidly evolved to “personal multitasking” systems
Process Control & Real-Time

- Computer is dedicated to a single purpose
- Classic embedded system
- Must respond to external stimuli in fixed time
- Continuous media popularizing real-time techniques
- An area of growing interest
Networks

- LAN (Local Area Network) evolution
- 3Mbps (1975) -> 10 Mbps (1980)->100 Mbps (1990)
- High speed communication means new way to do computing
  - Shared files
  - Shared memory
  - ???
Distributed OS

- Wave of the future

Multiple Computers connected by a Network
Evolution of Modern OS

- **Batch**
  - Memory Mgmt
  - Protection
  - Scheduling
  - Files
  - Devices

- **Timesharing**
  - Memory Mgmt
  - Scheduling
  - Protection

- **PC & Workstation**
  - System software
  - Human-Computer Interface

- **Network OS**
  - Client-Server Model
  - Protocols

- **Real-Time**
  - Scheduling

- **Modern OS**
Examples of Modern OS

• UNIX variants -- have evolved since 1970
• Windows NT -- has evolved since 1989 (much more modern than UNIX)
• Research OS -- still evolving …
• Book uses Linux as main example
• This course will use Nachos for the programming assignments
  – Supplementary materials available on the web
Microsoft Windows NT

- Heavily window-oriented
- Foundation behavior is windows-independent
Windows NT (cont)

- OS API has text orientation (like UNIX)
- Object-oriented implementation
- Heavy use of threads
- Broad spectrum of synchronization tools
- Modern I/O system