Operating Systems (ECS 150)
Spring 2011

Raju Pandey
Department of Computer Science
University of California, Davis
CA 95616
pandey@cs.ucdavis.edu
http://www.cs.ucdavis.edu/~pandey

Course Objectives

- After completing this course, you should have
  - broad understanding of components of modern high performance operating system, and theoretical issues associated with building operating system;
  - Implemented fragments of operating system components, polices, mechanisms, etc.; and
  - develop intuition for which system approaches work, and which don't;
- This course ….
  - is not about specific OS, say Linux or Windows XP, etc.
  - is not about APIs, standards, …
  - is more about OS concepts and their realization
  - We will use BSD primarily as an example OS concept
- Organizing Theme: OS Components and Issues
  - Kinds of components
  - Characteristics
  - Issues in building them

Administrative Matters

- Instructor:
  - Raju Pandey, pandey@cs.ucdavis.edu
  - 3041 Kemper Hall, 752-3584
  - Office Hrs: Tu/Th: 1:40 – 3:00 and with appointments
- TA:
  - Jesus Pulido (jpulido@ucdavis.edu)
  - Office hours: To be announced
- Details:
  - Lecture: T/Th 4:40 – 6:00 PM, 184 Young
- Communication
  - Discussion through smart site for the course
  - Course home page: accessible through `http://www.cs.ucdavis.edu/~pandey`

Administrative Matters - cont'd

- Textbook:
  - "The Design and Implementation of the FreeBSD Operating Systems" by Marshall Kirk, McKusick and George V. Neville-Neil (* Let’s wait a bit on this *)
  - Check course home page for other reference books
- Reading material
  - Text book
  - Manuals, HOW-TOS
  - FreeBSD Source code
- Copies of transparencies: Pick it up from course web site.
- Computing Resources:
  - CSIF Machines; Personal machines
  - More details forthcoming
- Software: Modifying FreeBSD

Course work

- Course load: Very high
- Project (40-45%)
- Homeworks (10-15%)
- Tests (40-50%)
  - Midterm (15-20%)
  - Final (20-30%)

Course work: Projects

- Projects: Implement OS concepts by
  1. extending/modifying a real operating system;
  2. rebuilding operating system;
  3. testing implementation by running applications on modified operating system
- Advice
  - Know C, if not brush up
  - Start to learn FreeBSD
- Submission details coming soon...

	Projects
I	Operating System Intro
II	System Call and Synchronization
III	Process management, scheduling
IV	Memory management
V	File system and I/O

Course work: Homework

- About 4 – 6; due in one week.
- Two parts:
  - Read text book sections and answer questions
  - Solve assigned problems
- Solutions will be made available
- Homework due in class; must submit before class starts.

Policies

- Regrades on homework
  - Must be done within one week of grading; Talk with TA first, followed by the instructor
- No makeup Midterm or Final examination.
- Final grade
  - Absolute grading.
  - Each homework, project, examinations given points that define A, B, C, D for each activity.
  - Final A, B, C, D computed by weighted average of these points
  - Your final graded weighted in a similar manner
  - Your final grade depends on where you fall..
- All work must be original; NO CHEATING.
  - More on this later..

Background

- Brush up on all within the first two weeks.
- C language:
  - Source files, include files
  - Macros: #define, #ifdef, #include, etc. + Preprocessors
  - static, extern, local and global functions and variables
  - int, char, float, void
  - Pointers; function pointers; address, *, &
  - Arrays, multi-dimensionals arrays, pointers as arrays, etc.
  - Memory model
- Shell: csh, tcsh, bourne, korn
  - Scripts, Environment variables, Utilities
- Compilation, linking, object files, libraries, shared libraries, dynamic libraries
- Tools: Editors, Compilers, linkers, make, gdb, tar/untar, zip/unzip/gzip
- Common operations: format and create floppies, mount and unmount directories, file permission, etc..

Scope of Course

- OS components
  - OS structures
  - Processes, threads
  - Memory management
  - File and I/O subsystems
  - Security
- Emphasis:
  - Core OS concepts
  - Design and implementation issues
  - Performance implications
  - Correctness and security implications

Syllabus

Tentative schedule:

Date	Topic
3/29	Introduction
3/31-4/5	Machine and OS Organization
4/7-4/12	Processes and Threads
4/14-4/19	Synchronization
4/21-2/26	Scheduling
4/28	Memory Management
5/3	*** Midterm*** (In class)
5/5-5/10	Virtual Memory
5/12-5/17	File System
5/19-5/24	I/O
5/26-/31	Security
6/2	Summary
6/4 (Saturday)	Final Exam: 1:00 PM – 3:00 PM

(* denote advanced topics that may be covered if there is time)

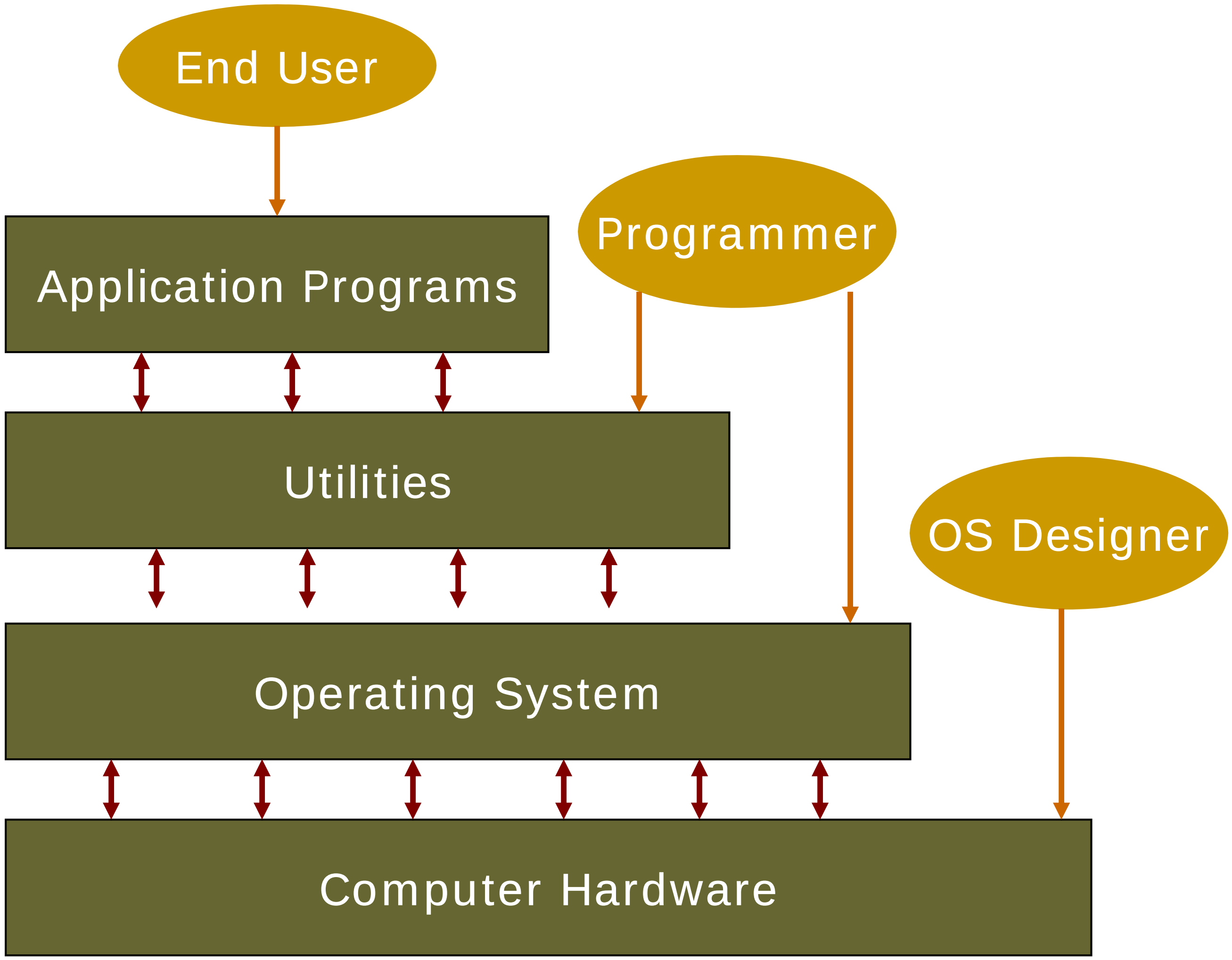
Overview

- What is an OS?
- What does an OS do?
- How is OS organized?
- How do we evaluate what an OS does?

Semantic Gaps

- Hardware capabilities at low level:
  - Low level operations on bits, bytes and words
  - Low level logical operations (gotos, conditional gotos)
  - Low level memory model (registers, raw memory words)
  - Asynchronous operation (timers, interrupts)
- Application semantics at a high level:
  - States represented as complex data structures
  - Units and collections of operations
  - Complex flow of operations
- Software used to provide mapping between high level and low level:
  - Language processors, linkers and loaders.
  - Language execution environments
  - **Operating Systems**

Semantic Gap and Software Layers



Semantic Gaps – cont'd.

- Machine instruction vs high level operation
  - Compiler
- Linear memory vs data structures
  - Compiler
- Limited Resources (CPU & memory) vs more needed
  - OS
  - Virtualization
- Secondary memory devices vs files
  - OS
- I/O devices vs high level I/O commands
  - OS

Introduction: Views of OSs

- An *extended machine*
  - Principle of **abstraction hides complexity**
  - OS provides high level operations using lower level operations
    - An interface between applications and hardware
    - Almost like a library, except that sometimes it intervenes without being explicitly called.
- A *virtual machine*
  - Principle of **virtualization supports sharing**
  - OS provides virtual CPU, memory, devices
- A *resource manager:* Abstract hardware resources (CPU, memory, persistent storage, network, etc.)
  - Control access to resources
  - **Balance** overall **performance with** individual **needs** (response time, deadlines)

Why OS? Objectives

- Programming simplicity
  - High Level API ->
  - Programming Model
  - Utilities
- Portability across different machine architectures
- User Benefits:
  - Safety
  - Fairness
  - Efficiency
- Ability to evolve

Major OS Issues

- Software engineering Issue:
  - How is OS organized? How are different components defined? What do they do? How do they talk with each other?
  - How can new features be added to it?
- Abstraction/Modeling Issues:
  - How are resources named?
  - How do OS and application components discover each other? How do they talk with each other?
  - How are parallel activities created and controlled?
  - How do we make data last longer than program executions?
  - How do multiple computers interact with each other?

*UW

Major OS Issues

- Resource Management issues:
  - How are resources shared?
  - How do we make things go faster?
  - What happens as demands and resources increase?
  - Accounting
- Security/Protection/Reliability issues:
  - What if something goes wrong?
  - How to protect one program from another?
  - How to ensure integrity of OS and its resources?
  - How to ensure access control?

*UW

Services Provided by OS

- Program development
  - Editors and debuggers
- Program execution
- Access to I/O devices
- Controlled access to files
- System access

Services Provided by OS – cont'd.

- **Error detection and response**
  - internal and external hardware errors
    - memory error
    - device failure
  - software errors
    - arithmetic overflow
    - access forbidden memory locations
  - operating system cannot grant request of application
- **Accounting**
  - collect statistics
  - monitor performance
  - used to anticipate future enhancements
  - used for billing users

Some things operating systems do

- Program management (Processes)
- Memory Management
- Scheduling / Resource management
- Communication
- Protection and Security
- File Management - I/O
- Naming
- Synchronization
- User Interface

Processes

- A unit of activity characterized by a single sequential thread of execution, a current state, and an associated set of system resources
- Three components:
  - Program
  - Associated data needed by the program
  - Execution context of the program
- Basis for
  - Scheduling
  - Resource management
  - Protection, access control
  - Accounting
- Variations:
  - Threads, Events

Process: Issues

- Mechanisms
  - Processes, Lightweight process, threads, events
  - System-Level, User-Level?
  - Machine-specific, Portable
  - Interaction with OS, User and Machine abstractions
- Cost
  - Context switching
  - Management cost
  - Concurrency
- Scheduling
  - Fairness
  - Guarantees
  - Real-time and software real-time constraints

Memory Management

- Process isolation
  - Safety
- Automatic allocation and management
  - Virtual Memory
  - Distributed shared memory
- Protection and access control
- Long-term storage
- Support for modular programming

Memory Management

- Mechanisms:
  - Memory Hierarchy
  - Single and mult-host memory models:
    - consistency, synchronization
  - Applications
  - Interaction with hardware
  - Recovery, Persistence
- Cost:
  - Page faults
  - Caching and replacement

Communication

- Interaction between processes
  - at local or remote nodes
- Information transfer
- Mechanisms
  - Shared memory, sockets, pipes, files, signals, interrupts
  - RPC, RMI
  - Group communications (One-one, one-many, many-one, many-many
  - Protocols
- Cost and performance
  - Latency, Scalability, Quality of Service

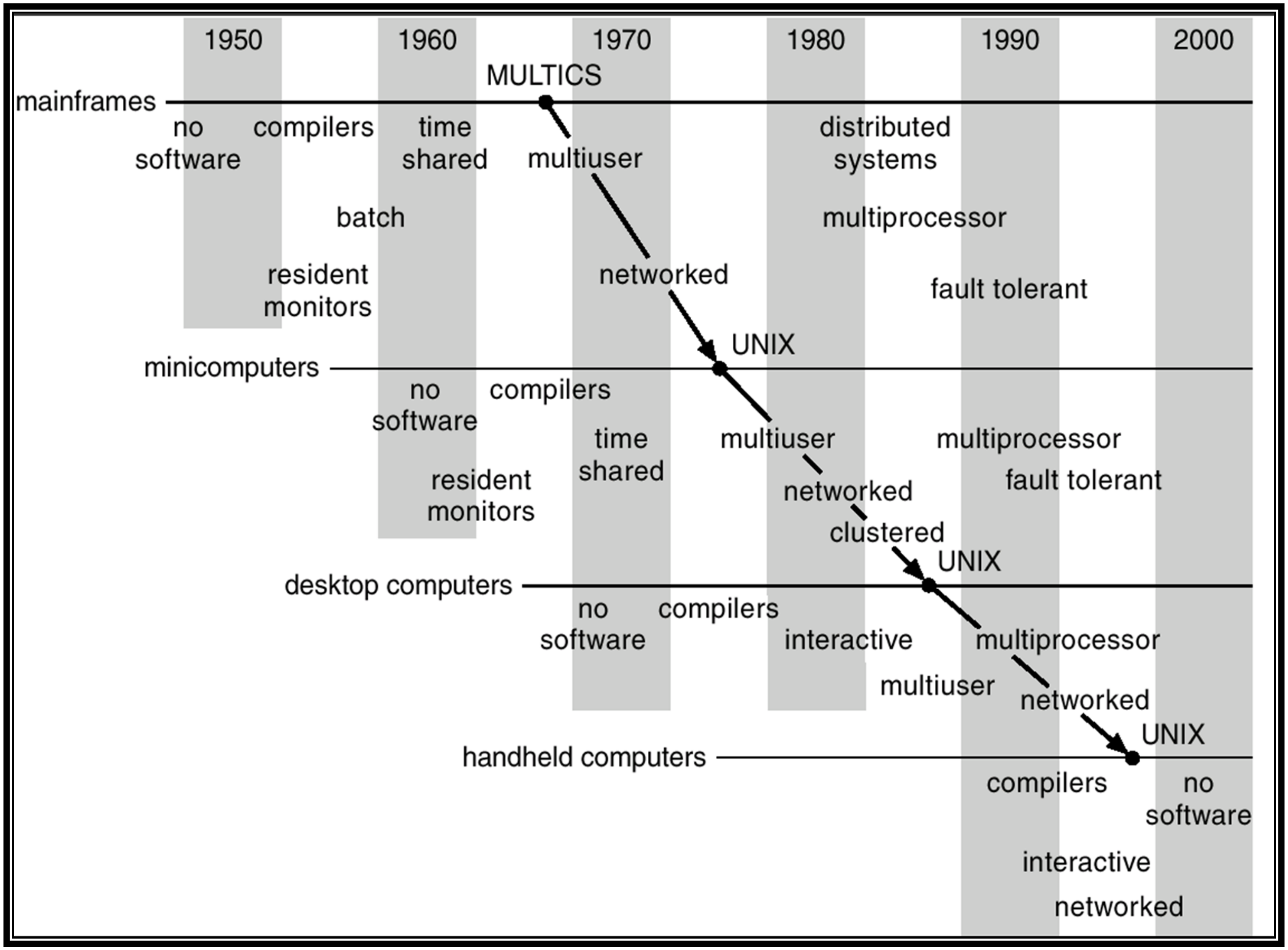
File and I/O Systems

- Long term archival
- Mechanisms and characteristics
  - File and I/O system models
  - Transparency
  - Consistency
- Algorithms:
  - Buffering
  - Data partitioning and placement
  - Scalability
- Performance:
  - Latency
  - Resource usage
  - Accessibility

Evolution of Operating Systems

- Dedicated machines
- Batch Processing
- Time Sharing
- Workstations and PC's
- Distributed Systems

Evolution of OS Concepts and Features



Evolution of OSs

- Serial Processing
  - No operating system
  - Machines run from a console with display lights and toggle switches, input device, and printer
  - Setup included loading the compiler, source program, saving compiled program, and loading and linking
- Simple Batch System:
  - Monitor: software that controls the running programs
    - Batch jobs together
    - Program branches back to monitor when finished
    - Resident monitor is in main memory and available for execution
    - Job control language for instruction to the monitor
  - Memory protection: do not allow the memory area containing the monitor to be altered
  - Timer: prevents a job from monopolizing the system

Evolution of OSs

- Multiprogramming Systems
  - Overlap CPU and I/O
  - Protection
  - Synchronization and Communication
  - Dynamic Memory Management (swapping and paging)
- Interactive OSs
  - Guaranteed response time
  - Time-sharing (quantum)

OS Evolution and Concepts

- PC and workstation OSs
  - GUI
- Real-time OSs
  - Deadlines (scheduling)
- Distributed OSs
  - Loosely coupled/tightly coupled
  - Consistent timeline (logical clocks, time stamps)
- Special Purpose OSs
  - Real-time OS
  - Embedded systems
  - Active routers