

CS 422S

Operating Systems Organization: Introduction

Fred Kuhns
fredk@cse.wustl.edu
Applied Research Laboratory,
Department of Computer Science and Engineering,
Washington University in St. Louis



Operating Systems Organization

- **Instructor:** *Fred Kuhns*
Phone: *935-6598*
Email: *fredk@cse.wustl.edu*
Office Hours: *Bryan 411*
Monday/Wednesday 4:00-5:00
- **Location:** *Lopata Room 101*
Times: *Monday/Wednesday 2:30 - 4:00*
Newsgroup: *wu.cs.class.422*
Web: *<http://www.cse.wustl.edu/~fredk/Courses/cs422>*

CS422S TAs

- **Xiaofeng Chen**
Email: *chenfrank_98@yahoo.com*
Office: *Lopata 408*
Office Hours: *TBD*
- **Christopher King**
Email: *cck1@cec.wustl.edu*
Office: *Lopata 408*
Office Hours: *TBD*
- **Philip Wang**
Email: *plw4@cec.wustl.edu*
Office: *Lopata 408*
Office Hours: *TBD*

Course Materials

- **Textbooks**
 - Required: Gary Nutt, *Operating Systems: A Modern Perspective, Second Edition, Lab Update*, Addison Wesley, 2002
 - Desirable: Gregory Andrews, *Foundations of Multithreaded, Parallel, and Distributed Programming*, Addison Wesley, 2000
- **Other useful references:**
 - *UNIX Network Programming*, 2nd edition, Volume 2, W. Richard Stevens, Prentice Hall, 1998.
 - *The C Programming Language: Second Edition*, Kernighan and Ritchie, Prentice Hall, 1991
 - *The Practice of Programming*, Brian Kernighan and Rob Pike, Addison-Wesley, 1999
 - Class handouts and presentation material
 - Man pages and Web accessible documentations

Course Work and Grading

- Exam - 40% of grade
 - mid-term 15%, final 25%
 - Final is comprehensive
- Homework and Quiz - 10% of grade
 - Homework problems or minor programming assignments.
 - Quizzes announced or unannounced
- 4 +/- Programming Projects - 50% of grade
 - Must use UNIX (Linux or Solaris as required by project description) and C.
- **Late Policy:** 5 points per day for up to 3 days. After 3 days assignments will not be accepted.
- **Email policy:** Please do not email questions (related to assignments) directly to myself or the TAs, instead post them to the newsgroup so all may benefit from the questions and answers. Of course I also encourage you to visit the TAs or myself during office hours.

Grading Projects

- **Design 30%**
 - Are all conditions covered
 - Have synchronization and scheduling issues been addressed
 - Used proper measurement and/or data collection techniques
 - Does it demonstrate and understanding of the underlying issues
 - Does solution demonstrate insight or have novel approaches been used
- **Structure 20%**
 - Logically structured
 - Understandability (comments, names etc)
 - Maintainability or portability issues
- **Documentation 30%**
 - Must include a README file (ascii, <= 80 columns/line, UNIX format)
 - Does analysis address all required issues and is it insightful
 - Logically structured and understandable
 - Followed instructions
- **Operation 20 %**
 - Work are specified or does it terminate in error
 - Accept required parameters
 - Results displayed properly
 - Results correct or reasonable

Syllabus

- Introduction and overview of the C programming language
- Top down view of Operating Systems and System Software
- Overview of computer architecture, devices and device management
- Processes and threads
- CPU Scheduling (uni-, multi- and RT)
- Concurrency Issues
- Memory management and Virtual Memory
- File systems, I/O and secondary storage
- Network I/O (sockets and streams)
- Distributed Processing
- Security (if time permits)

Computer Systems

- Hardware and software combine to solve specific problems
 - Software solves some problem: entertainment, information management, scientific problem solving, system control, etc.
 - Hardware provides the basic computing resources (Processor(s), Memory, System Bus and I/O modules). Although strictly speaking many solutions can be designed and implemented entirely in hardware but this is a course on operating system.
- Software is divided into two categories:
 - **Application software** - used directly by user (person or possibly another program or computer system) to solve some problem. Ultimately a computer system exists to implement applications.
 - **System software** - think of this as convenience software: it simplifies application programming by creating a convenient to use programming environment. It does this implementing the low level hardware management functions and providing a standardized (we hope) interface and set of abstractions. For example, if you write a C++ program you wouldn't want to also implement the interrupt service routines, filesystems, compilers, linkers etc..

Goals of an OS

- **User Environment** - OS layer transforms bare hardware machine into higher level abstractions
 - **Execution environment** - process management, file manipulation, interrupt handling, I/O operations, language.
 - **Error detection and handling**
 - **Protection and security**
 - **Fault tolerance and failure recovery**
- **Resource Management**: an OS is asked to be efficient, transparent and often feature rich - no small task!
 - **Time management** - CPU and disk transfer scheduling
 - **Space management** - main and secondary storage allocation
 - **Synchronization and deadlock handling** - IPC, critical section, coordination
 - **Accounting and status information** - resource usage tracking

Driving Concepts

- **Abstraction:** resource abstraction used to manage complexity. Low-level computing resources (the hardware interface) is abstracted to create an *extended machine*.
- **Virtualization:** Resources are virtualized to permit controlled sharing and isolation. An OS is expected to transparently share resources among concurrently executing programs.
- **Resource management:** OS ensures applications make progress by providing policies and mechanisms that enable both implicit and explicit allocation. Must not impose undue performance penalties while *maximizing utilization, minimizing overhead*, providing *fair/predictable access to system resources*.

Resource Abstraction

- A computer system is composed of a collection of hardware components, AKA resources. To gain the greatest degree of flexibility and reduce dependence on specific HW architectures OSes abstract the notion of a resource to include software based components such as files, sockets or mailboxes.
 - Resource := an abstraction that *represents any hardware or software component* that may be used by an application
 - Resource Abstraction: provide an abstract model of hardware components and their operation. Generally desire a common model across similar resource types.
 - OS designers goal is to select abstraction that provide the greatest flexibility while providing a general interface that is convenient to use.
 - Over generalization may result in an inflexible implementation.
- Resource Examples: Disk drive
 - a disk is composed of platters, heads, sectors, tracks and blocks
 - applications deal with abstract files
 - operating system provides the abstract model and behavior

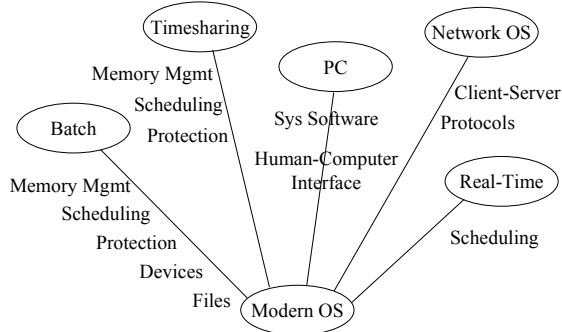
Resource Sharing

- If a system support the concurrent execution of multiple programs (aka processes) then it must provide mechanisms and policies for sharing system resources
 - concurrency apparent or true, the later case being known as parallel execution.
 - abstract machines *transparently* share resources, the OS implements a specific allocation policy to provide isolation - the OS is considered trusted software.
 - concurrent programs may also elect to *explicitly* share resources
 - Resource Sharing may be *Space-multiplexed* or *Time-multiplexed*
- OS implements
 - fundamental abstraction of hardware components and mechanisms to manage sharing
- Examples:
 - loading multiple programs into physical memory. One program can not read or write the memory locations of another process - *protection and isolation*
 - A program may explicitly permit another program to access part of its memory for efficient inter-program communication

Historical Perspective

- *Batch Processing* - multiprogramming, batch of jobs, no user interaction, optimize resource utilization
- *Timesharing* - multiprogramming, interactive, many processes, resource isolation, optimize response time
- *Personal Computing* - single user multiprogramming environment, graphics, multimedia, small OS
- *Process control and real-time* - dedicated to single task, embedded systems, timing constraints, hard versus soft
- *Network OS* - communication across network, local versus remote resource management.
- *Distributed OS* - transparency

Modern OS Evolution



OS Strategies

- ✓ **Batch processing:** non-interactive jobs with throughput primary concern (max jobs/s)
- ✓ **Timesharing:** multiple interactive jobs, utilizes multiprogramming, abstract (virtual) machine concept, protection, security, optimize response time.
- ✓ **Personal computers and workstations:** commodity hardware and software, interoperability, dedicated systems
- ✓ **Embedded:** limited resources (power, memory, processor speed), interact with other devices
- ✓ **Real-time:** timing constraints on computations, desire deterministic behavior, scheduling algorithms
- ✓ **Small communicating computers:** form of embedded system but may be more severe restrictions, multimedia common (streaming media), non-traditional platforms

General Purpose Operating Systems

- We will spend the majority of the semester studying what is classically known as a general purpose operating system.
- Typified by desk-top and portable computing systems.
- Most mechanisms we cover will equally apply to all types of operating systems
- We will also look at policies and mechanisms commonly attributed to distributed and real-time operating systems

Distributed Operating Systems

- Controls and manages resources for a network of autonomous computers
 - manage both hardware and software resources
 - behaves as a single monolithic system.
- User not aware of program or resource location
- Design issues same as traditional systems
- Practical issues:
 - lack of shared memory
 - lack of global clock (i.e. time reference)
 - unpredictable communication delays.

Multiprocessor Operating Systems

- Consists of a set of processors that
 - share a set of physical memory blocks
 - share a common clock
 - "share" over an interconnection network.
- Control and manage resources
 - hardware and software resources
 - viewed as a uniprocessor system.
- Design issues same as traditional system.
- Practical issues:
 - increased complexity of synchronization, scheduling, memory management, protection and security.

Database Operating Systems

- Database systems place increased demands on an operating system to efficiently support:
 - concept of a transaction
 - manage large volumes of data
 - concurrency control
 - system failure control

Real-time Operating Systems

- Place application specific special requirements on an operating system.
- Policies and mechanisms are geared to ensuring jobs meet their deadlines.
- Problem is one of resource scheduling and overall system utilization.
