

Distributed and Operating Systems

Spring 2013

Prashant Shenoy

UMass Computer Science

<http://lass.cs.umass.edu/~shenoy/courses/677>



Course Syllabus

- CMPSCI 677: Distributed and Operating Systems
- *Instructor:* Prashant Shenoy
 - Email: shenoy@cs.umass.edu, Phone: (413) 577 0850
 - Office hours: Monday 10:20-11:30, CS 336, or by appt
- *Teaching Asst:* Manjunath Narayana
 - Email: narayana@cs.umass.edu, Phone: TBD
 - Office hours: TBD
- *Course web page:* <http://lass.cs.umass.edu/~shenoy/courses/677>



Course Outline

- Introduction (*today*)
 - What, why, why not?
 - Basics
- Distributed Architectures
- Interprocess Communication
 - RPCs, RMI, message- and stream-oriented communication
- Processes and their scheduling
 - Thread/process scheduling, code/process migration, virtualization
- Naming and location management
 - Entities, addresses, access points



Course Outline

- Canonical problems and solutions
 - Mutual exclusion, leader election, clock synchronization, ...
- Resource sharing, replication and consistency
 - DFS, consistency issues, caching and replication
- Fault-tolerance
- Security in distributed Systems
- Distributed middleware
- Advanced topics: web, cloud computing, green computing, multimedia, and mobile systems



Misc. Course Details

- *Textbook*: Distributed Systems, 2nd ed, by Tannenbaum and Van Steen, Prentice Hall 2007 (recommended)
- *Grading*
 - 4-5 Homeworks (15%), 3-4 programming assignments (40%)
 - 1 mid-term and 1 final (40%), class participation + quizzes (5%)
- *Course mailing list*: cs677@cs.umass.edu
 - Spire will automatically add you to this list.
- *Pre-requisites*
 - Undergrad course in operating systems
 - *Good* programming skills in a high-level prog. language



Why Distributed Systems?

- Many systems that we use on a daily basis are distributed
 - World wide web, Google
 - Amazon.com
 - Peer-to-peer file sharing systems
 - SETI@Home
 - Grid and cluster computing
 - Modern networked computers
- Useful to understand how such real-world systems work
- Course covers basic principles for designing distributed systems



Definition of a Distributed System

- A distributed system:
 - Multiple connected CPUs working together
 - A collection of independent computers that appears to its users as a single coherent system
- Examples: parallel machines, networked machines



Advantages and Disadvantages

- Advantages
 - Communication and resource sharing possible
 - Economics – price-performance ratio
 - Reliability, scalability
 - Potential for incremental growth
- Disadvantages
 - Distribution-aware PLs, OSs and applications
 - Network connectivity essential
 - Security and privacy



Transparency in a Distributed System

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource may be shared by several competitive users
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

Different forms of transparency in a distributed system.



Open Distributed Systems

- Offer services that are described a priori
 - Syntax and semantics are known via protocols
- Services specified via interfaces
- Benefits
 - Interoperability
 - Portability
- Extensibility
 - Open system evolve over time and should be extensible to accommodate new functionality.
 - Separate policy from mechanism



Scalability Problems

Concept	Example
Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized algorithms	Doing routing based on complete information

Examples of scalability limitations.



Scaling Techniques

- *Principles* for good decentralized algorithms
 - No machine has complete state
 - Make decision based on local information
 - A single failure does not bring down the system
 - No global clock
- *Techniques*
 - Asynchronous communication
 - Distribution
 - Caching and replication



Distributed Systems Models

- Minicomputer model (e.g., early networks)
 - Each user has local machine
 - Local processing but can fetch remote data (files, databases)
- Workstation model (e.g., Sprite)
 - Processing can also migrate
- Client-server Model (e.g., V system, world wide web)
 - User has local workstation
 - Powerful workstations serve as servers (file, print, DB servers)
- Processor pool model (e.g., Amoeba, Plan 9)
 - Terminals are Xterms or diskless terminals
 - Pool of backend processors handle processing



Distributed System Models (contd)

- Cluster computing systems / Data centers
 - LAN with a cluster of servers + storage
 - Linux, Mosix, ..
 - Used by distributed web servers, scientific applications, enterprise applications
- Grid computing systems
 - Cluster of machines connected over a WAN
 - SETI @ home
- WAN-based clusters / distributed data centers
 - Google, Amazon, ...
- Virtualization and data center
- Cloud Computing



Emerging Models

- Distributed Pervasive Systems
 - “smaller” nodes with networking capabilities
 - Computing is “everywhere”
 - Home networks: TiVO, Windows Media Center, ...
 - Mobile computing: smart phones, iPods, Car-based PCs
 - Sensor networks
 - Health-care: personal area networks
- Sustainability as a design goal



Uniprocessor Operating Systems

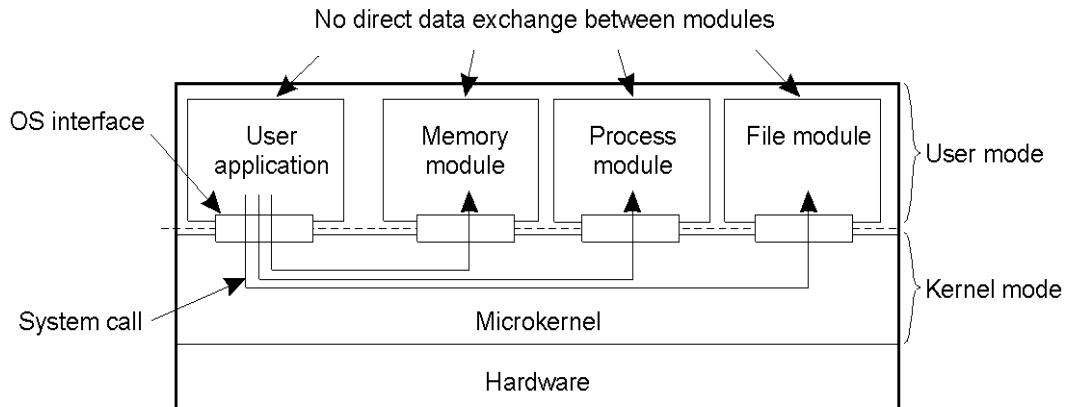
- An OS acts as a resource manager or an arbitrator
 - Manages CPU, I/O devices, memory
- OS provides a virtual interface that is easier to use than hardware
- Structure of uniprocessor operating systems
 - Monolithic (e.g., MS-DOS, early UNIX)
 - One large kernel that handles everything
 - Layered design
 - Functionality is decomposed into N layers
 - Each layer uses services of layer N-1 and implements new service(s) for layer N+1



Uniprocessor Operating Systems

Microkernel architecture

- Small kernel
- user-level servers implement additional functionality



Distributed Operating System

- Manages resources in a distributed system
 - Seamlessly and transparently to the user
- Looks to the user like a centralized OS
 - But operates on multiple independent CPUs
- Provides transparency
 - Location, migration, concurrency, replication,...
- Presents users with a virtual uniprocessor



Types of Distributed OSs

System	Description	Main Goal
DOS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency



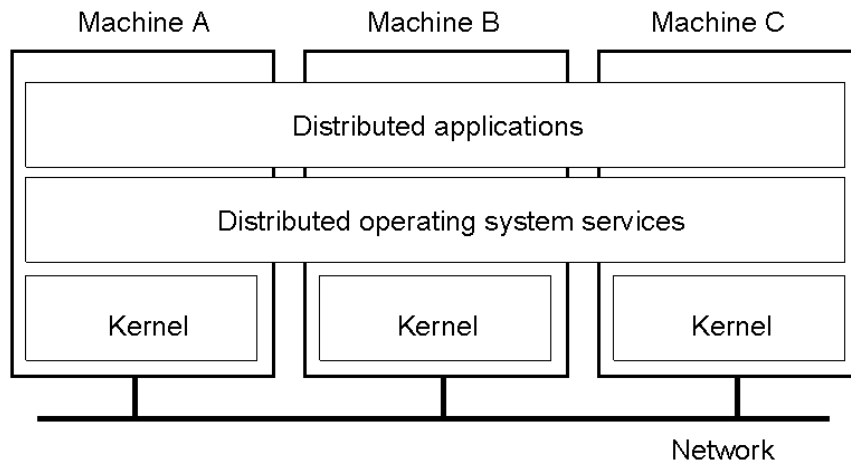
Multiprocessor Operating Systems

- Like a uniprocessor operating system
- Manages multiple CPUs transparently to the user
- Each processor has its own hardware cache
 - Maintain consistency of cached data

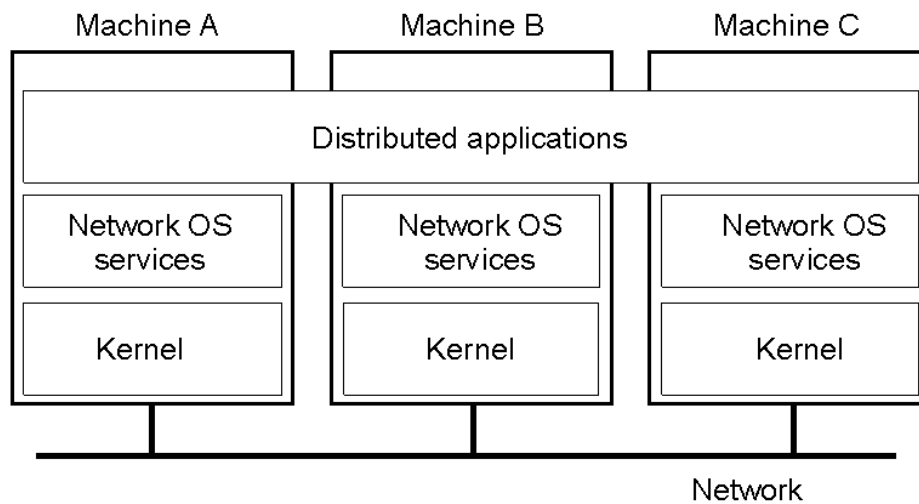


Multicomputer Operating Systems

Example: MOSIX cluster - single system image

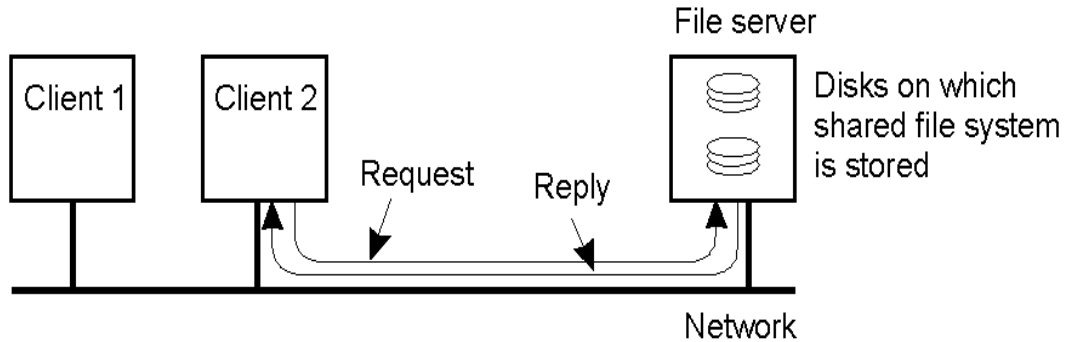


Network Operating System



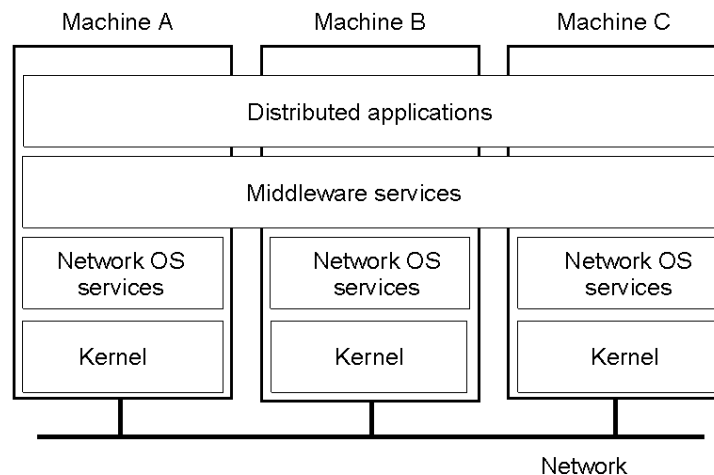
Network Operating System

- Employs a client-server model
 - Minimal OS kernel
 - Additional functionality as user processes



Middleware-based Systems

- General structure of a distributed system as middleware.



Comparison between Systems

Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open

