Distributed Operating Systems

Ing. Tomáš Seidmann, PhD.
Faculty of Informatics and Information Technology
Slovak University of Technology in Bratislava
seidmann@fiit.stuba.sk
http://www.cdot.ch/thomas/

– Typeset by FoilTEX –

Course outline

This course focuses on software issues in the design and implementation of modern computer systems, particularly the operating systems and distributed algorithms that are essential in supporting networking and distributed processing.

- Fundamental concepts (transparency, service, coordination)
- Distributed processes (synchronization, communication and scheduling)
  - Concurrent processes and programming
  - Process interaction
  - Process scheduling
- Distributed resources (files and memory)
  - Distributed file systems
  - Distributed shared memory
  - Security issues in network and distributed environments

References

[1] Andrew S. Tanenbaum and Martin van Steen, Distributed Systems: Principles and Paradigms (2nd Edition), Prentice Hall 2007, ISBN 0-132-39227-5

[2] Randy Chow and Theodore Johnson, Distributed Operating Systems and Algorithms, Adison-Wesley 1997, ISBN 0-201-49838-3

Operating System Fundamentals

Functionality of operating systems can be divided into two categories:

- system services
- kernel (nucleus)

View on operating systems:

- Machine abstraction (extended machine), primary design goal
- Resource manager (means of achieving the goal)

Evolution of operating systems:

- Centralized operating systems (conventional) - well understood
- Network operating systems and distributed operating systems - caused by proliferation of personal workstations and LANs
- Cooperative autonomous systems - emerging concept: Computational Grids

Evolution of modern operating systems

Generation	System	Characteristics	Goals
first	centralized operating system	process management memory management I/O management file management	resource management extended machine (virtuality)
second	network operating system	remote access information exchange network browsing	resource sharing (interoperability)
third	distributed operating system	global view of : file system, name space, time, security, computational power	single computer view of multiple computer systems (transparency)
fourth	cooperative autonomous system	open and cooperative distributed applications	cooperative work (autonomicity)

Centralized operating systems

Structuring the OS software in manageable modules:

- vertical - layering
- horizontal - partitioning within layers
- enhancing the portability - separating the hardware-dependent code from the system - **minimal kernel** approach, which is typically **monolithic**.

Centralized operating systems II

Typical functions of a minimal kernel:

- multiplexing of processors with multiprogramming support
- interrupt handling
- device drivers
- process sychronization primitives
- interprocess communication primitives

A universal minimal kernel on which standard operating systems can be implemented to support application-oriented subsystems is called a *microkernel*.

OS's function as a resource manager:

- mapping (scheduling) of processors - **multiprogramming** and **timesharing**
- **process synchronization** and **interprocess communication**
- process **scheduling**
- management of I/O operations, enhancing with **spooling** and **buffering**
- providing **virtual memory**
- management of **files**

Network operating systems

Computer network = loosely coupled multiple computer systems where no direct hardware or software control of one workstation to another exists.

- straightforward extension of a traditional operating system to facilitate resource sharing and information exchange
- information exchange divided and implemented at various levels - *communication subnetwork* to *transport services*
- high-level API for transport services, such as *sockets* and *remote procedure call*
- NOS characterized by inclusion of a transport layer and the support for network applications like:
  - remote login
  - file transfer
  - messaging
  - network browsing
  - remote execution

Distributed operating systems

Sharing of resources and coordination of distributed activities in networked environments are the main goals in the design of a distributed operating system. The key distinction between a network OS and a distributed OS is the concept of *transparency*:

- concurrency transparency (also in centralized OS)
- location transparency
- parallelism and performance transparency
- migration transparency
- replication transparency

Distributed operating systems consist of three major components:

- coordination of distributed processes
- management of distributed resources
- implementation of distributed algorithms

Cooperative autonomous systems

- distributed systems are characterized by service decomposition
- cooperative autonomous systems emphasize service integration

The emerging need for cooperative autonomous systems has triggered several standardization efforts for development of distributed software:

- Common Object Request Broker Architecture (CORBA)
- Java RMI/JNDI/Jini, .NET Remoting
- Web Services/UDDI
- WCF(Indigo, .NET 3.0), SCA (Java, C++, PHP...)

Important features:

- use of intelligent *trader* or *broker* architecture
- can be viewed as a *software bus*
- serve as *middleware* that supports distributed cooperative applications

Distributed algorithms

Distributed systems have following specialties compared to centralized systems:

- **Message passing** as the media of all coordination among concurrent processes due to the lack of shared memory. The distributed algorithms may be fully decentralized or centralized (in the latter case a distributed election algorithm is required).
- **Lack of global information** due to network delays and unreliable system components. Includes the lack of global timing.
- **Data replication.** The primary goal of the protocols is to maintain data consistency.
- **Failures and recovery**. Fault tolerance becomes more critical issue for distributed systems. It uses redundancy of resources and services. Recovery is a passive approach in which the state of the system is maintained and used to roll back the execution to some checkpoint.

Distributed system concepts and architectures

Goals:

- **Efficiency** is more complex in distributed systems than in centralized systems due to the effect of communication delays. With respect to system load distribution, problems such as bottlenecks and congestion either in the physical networks or software components must be addressed. Computation speed and system throughput can be enhanced through distributed processing and load sharing if the communication system is carefully designed.
- **Flexibility** includes the *friendliness* of the system and the *freedom* of the user in using the system - ease of use of the system interface and the ability to relate the computation processes in the user's problem domain to the system. The object-oriented strategy is a commonly used strategy in achieving this goal. From the system's view flexibility is the system's ability to evolve and migrate - modularity, scalability, portability and interoperability.
- **Consistency** is more difficult to achieve in a distributed system due to the lack of global information, potential replication and partitioning of data, the possibility of component failures and the complexity of interaction among modules. The system must be capable of maintaining its integrity with proper concurrency control mechanisms and failure handling and recovery procedures.
- **Robustness**. Failures (in communication links, processing nodes and client/server processes) are more frequent than in a centralized single computer system. Meaning of robustness:
  - fault tolerance: ability to reinitialize itself to a consistent state with only some possible degradation of its performance
  - security

Transparency

- Goal motivated by the desire to hide all irrelevant system-dependent details from the user, whenever possible.
- It is more important in distributed systems due to higher implementation complexities.
- Shielding the system-dependent information from the users is a trade-off between simplicity and effectiveness.

- **Access transparency** - accessing both local and remote system objects in a uniform way.
- **Location transparency** - no awareness of object locations. Sometimes called **name transparency**.
- **Migration transparency** - ability to move an object to a different location without changing its name; also called **location independence**.
- **Concurrency transparency** - allow the sharing of objects without interference.
- **Replication transparency** - consistency of multiple instances (or partitioning) of files and data.
- **Parallelism transparency** - parallel activities without users knowing how, when and where.
- **Failure transparency** - fault tolerance.
- **Performance transparency** - attempts to achieve a consistent and predictable performance level even with changes of the system structure or load distribution.
- **Size transparency** - modularity and scalability.
- **Revision transparency** - vertical growth of the system.

Services

- An operating system is a service provider.
- There are different levels of services.
- The most fundamental services are implemented in the kernel of each node of the system: **system primitives**.
  - communication
  - synchronization
  - processor multiplexing
  - **send** and **receive** primitives for message passing (asynchronous or synchronous).
- Functions basic to the operation of a distributed system, but implemented anywhere: **system servers**.
  - **name server** or **directory server**
  - **network server** - transformation of addresses of locations to communication paths and routing information
  - **time server** - physical and logical clocks
  - **file** and **print servers**
  - **migration server**
  - **authentication server**
- Higher-level or special-purpose services: **value-added servers**.
  - **group server** - groups of interacting processes
  - **distributed conferencing server**
  - **concurrent editing server**

Architecture models

Distributed system architectures:

- **workstation-server** model: a workstation may serve as a stand-alone computer or as a part of an overall network.
- **processor-pool** model: collecting all processing power in one place and leaving the users with only a terminal (added parallelism transparency).

Communication network architecture:

- point-to-point
- multipoint
  - bus-based: IEEE 802 LAN standards
  - switched: ISDN, SMDS, ATM
- for distributed systems the ratio of propagation delay to transmission delay is important: smaller values means more "closeness" of the system components (more suitable for systems requiring interactive exchanges of shorter messages), larger values lead to a more "communication oriented" system.

Communication network protocols

- Sets of rules that regulate the exchange of messages to provide a reliable and orderly flow of information among communicating processes.
- Two categories of communication services:
  - connection-oriented, network level: virtual circuit, communication hardware level: circuit switching
  - connectionless, network level: datagram, communication hardware level: packet switching
- Communication hardware and software is normally structured in layers.
- Standardized network specification are called **network system architectures**.
- Layers for a standardized network architecture are referred to as a **protocol suite**.
- Two of the most popular network protocol suites: ISO **OSI** and U.S. DoD **TCP/IP**.

Major design issues

A distributed system consists of concurrent processes accessing distributed resources (which may be shared or replicated) through message passing in a network environment that may be unreliable and contain untrusted components. This raises many design and implementation issues, in particular how to support transparency.

- **Object models and naming services** Objects in a computer system are processes, data files, memory, devices, processors, networks. It is tempting to assume that all objects can be represented in a uniform way - objects are encapsulated in servers and the only visible entities in the system are the servers. Servers must be identifiable:
  - Identification by name (name server);
  - Identification by physical or logical address (network server);
  - Identification by service that the servers provide - critical to the implementation of autonomous cooperating systems.
- **Distributed coordination**
  - Barrier synchronization
  - Condition coordination
  - Mutual exclusion
  - Deadlock detection/prevention

  The task of coordination is complicated by the fact of absence of global state: distributed resolution protocols or a centralized coordinator are the solutions. The role of the centralized coordinator can be moved from one process to another so that the coordinator will not become the central point of failure. Lack of global state can be circumvented with agreement protocols: message passing algorithms that achieve consensus in a distributed system with potentially faulty components.

- **Interprocess communication** It is desirable to have transparency in communication by providing higher-level logical communication methods that hide the physical details of message passing:
  - The client/server model;
  - Remote procedure call (RPC);
  - Group management and group communication (cornerstone of CSCW - Computer Supported Cooperative Work).
- **Distributed resources**
  - load distribution
    - multiprocessor scheduling
    - load sharing
  - distributed shared memory
  - distributed file systems
- **Fault tolerance and security**
  - Security threats and failures are both system faults.
  - The problem of failures can be alleviated if there is redundancy in the system - problem with checkpointing.
  - Security: authentication and authorization.