CMPSCI 677 Operating Systems **Spring 2015**

Lecture 1: January 22

Lecturer: Prashant Shenoy *Scribe: Amee Trivedi*

1.1 Course Introduction

The course Operating Systems is taught by professor Prashant Shenoy, in CMPSCI 142, every Monday and Wednesday from 9:05am to 10:20am. For all course information (syllabus, additional readings, lecture video, notes, mailing list, etc) visit `http://lass.cs.umass.edu/~shenoy/courses/677/`.

1.1.1 Grading

Student's grades will consist of 4-5 Homeworks (15%), 3-4 programming assignments (40%), 1 mid-term and 1 final exam, which would be a take home, (a total of 40%). Additionally, class participation and quizzes amount to a total of 5%.

It is expected that students already have a *good* set of programming skills in some high-level programming language.

1.2 Introduction to Distributed Systems

A distributed system can be broadly defined as a collection of independent computers that appear as a single system to the user. This broad definition includes parallel machines as well as networked machines. The main difference between parallel machines and networked machines is that in a parallel system multiple CPU's constitute a single machine and they are interconnected via a system bus. Whereas, in a networked machine multiple independent machines are interconnected via an ethernet cable or WAN.

Most of the systems that we use today are distributed in nature. Be it searching on google, checking mails, sharing files or systems like SETI@Home, grid, cluster computing systems.

1.2.1 Advantages

A distributed system has multiple advantages over centralized system. Few of them are as below:

- Communication and Resource Sharing: The connected nature enables *communication* both for users (high-level) and for systems (low-level). With this ease of communication comes the ability to *share resources* such as data, memory, computational resources, etc. One very common example of this kind of sharing is network file systems which allows distributed systems to store and retrieve files.
- Economics: Sharing resources and harnessing the power of multiple machines results in a lower price/performance ratio. Example it is better to have multiple cheap CPU's /machines connected to form a powerful unit rather than buy an expensive main frame to perform a task.

- Reliability and Scalability: Having multiple systems all working with a single purpose allows for greater *reliability* and *scalability* if the distributed system is designed well. Being well designed means that failures are expected and redundancy is built in. A single failure shouldn't result in a failure of the distributed system as a whole. Failures are more common because the probability of a single machine failing is less than the probability that one of many machines will fail.

- Incremental growth: A good distributed system design also entails being able to use more machines than are currently available for the same application. This allows your system to *grow incrementally* and add capacity as is needed.

1.2.2 Disadvantages

- Complexity: As a program becomes distributed in nature, the program complexity starts building up. The operating systems and applications themselves must also be distribution-aware.

- Requirement for Network connectivity: Since, a network connectivity is essential for communication, loss of connectivity is a major drawback

- Security and Privacy: Due to resource sharing and ease of communication possibilities of attacks and security become a concern.

1.2.3 Transparency

Transparency in distributed systems refers to what is hidden from the user of the system. It is desirable to have transparency in a distributed system because the more transparent the system is the easier it is for end user to use. There are many forms of transparency:

Access
The difference between the actual data and how it is accessed is hidden from the user.

Location
The user doesn't know where a resource is located.

Migration
The user doesn't know when a resource may move or is moved.

Relocation
A resource may move to another location while being used and the user doesn't know.

Replication/Concurrency
A resource may be shared by multiple competitive users and each user isn't aware of the others.

Failure
The user isn't aware of when failure and recovery occurs.

Persistence
A (software) resource may be in memory or on the disk and the user doesn't know.

1.2.4 Open Distributed Systems

Systems which offer services according to standard rules that describe the syntax and semantics of the services are *Open*. The syntax and semantics of open systems are published protocols that can be used via an interface (API). Hence, Open systems promote interoperability, portability and extensibility because they are intended to be used by a wide variety of users.

1.2.5 Scalability

The ability of a distributed system to run on a larger environment and scale is called property of scalability. Based on the design of the system, sometimes it is the need of the application to have centralized service, where a single server is used by all users. similarly there might be a centralized data or a centralized logic/algorithm need in a distributed system design. Such centralized items create bottle neck in the system. For example, you might have a centralized system for processing but with a distributed database. Another similar problem is that of *centralized algorithms*. This means that a computation requires complete information and so it can only be computed by a centralized system.

To avoid problems with centralized algorithms a set of Scaling techniques/principles for good decentralized algorithm design are laid down. There are 4 main rules as below:

- No machine should have the complete state: State means any data in a system needed for decision making.
- Make decisions based on local information
- A single failure doesn't cause a total system failure.
- No global clock

Techniques to avoid scalability problems include asynchronous communication, caching and replication.

1.3 A Quick History

The first distributed systems used early dialup communication (before the Internet). These systems were the *minicomputer model* where each user had a local machine for processing but remote data could be fetched. This led to the first E-Mail systems. Next the *workstation* model was developed in the 1990's to allow processing to also migrate (not just data). An example of this was the Sprite system which we will discuss later. Following the workstation model is the *client-server* model which is still very much used today. Users have local workstations and they connect to powerful workstations that serve as servers. These servers can be file, print, database, WWW servers (and more).

The *processor pool* model has developed more recently with systems such as Amoeba and Plan 9. In this model there are terminals (Xterms or diskless terminals) and a pool of backend processors that are available to perform processing. The processor pool could be made up of one or many individual servers. This is similar to the *cluster computing* used in data centers. Cluster computing is a local area network (LAN) with lots of servers and storage. Grid computing systems is essentially the same a cluster computing, but the machines are connected over a WAN such as the Internet. An example of Grid computing is SETI@home. This combined with virtualization and distributed data centers have led to the recent rise of cloud computing.

1.3.1 Emerging Models

Currently still developing is the idea of distributed pervasive systems which are made up of very small computing devices with networking capabilities. These devices include: smart phones, TiVO, Windows Media Center, car-based PCs, sensor networks, etc. The main idea is that computing is available everywhere.

1.4 Uniprocessor Operating Systems

These Operating systems were designed to operate on a single processor machine. All the current day OS have transitioned from Uniprocessor to Multiprocessor.There are a variety of ways to structure such an OS:

Monolithic There is one massive kernel that handles everything that the OS needs to do. This was used in early UNIX systems and MS-DOS.

Layered The required functionality is broken up into N layers. Each layer can only use services provided by the layer below it (N-1) to implement new services for the layer above it (N+1).

Microkernel The OS kernel is a small compact core and lot of functionalities are pulled put from the kernel and run as an application that runs in the user space. This design approach makes the kernel more modular, it is easy to make changes in the file system. The main drawback of this is decreased performance due to additional communication.