**Inet Sever coding experience**:

This was pretty straightforward and simple. I just typed it in and come up with no errors. After it was all in I began going through line by line and documenting it to the best of my knowledge. I had to look up a few things in the java 6 api on sun’s website. These were mostly minor calls and classes I was unfamiliar with working with. For instance I have never worked with the InetAddress class before. Everything worked out.

I double-checked the assignment right before the due date and then added extra header comments that I was unaware of. I found these by going more thoroughly through the course webpage.

**JokeServerClient coding experience**:

I began with a blank page and worked in things from the Inet server assignment that I figured would work correctly here. I began with very little functionality. Just to make sure every step I took was working. The first iteration had no admin client. It didn’t save in state. All it did was ask the client users name and then returned a single joke to that user.

A couple days later I came back and added five random jokes. This did not seem difficult at first. I stored the jokes in an array. How do I work in the users name when calling a random number?

Global variables are my friend. Since I couldn’t make anymore classes then the 3 required (the object oriented side of my brain hated this) I had to do all the state handling inside the worker thread classes. This ultimately wasn’t too bad, but it did make for a big pile of messy code.

StringTokenizer class saved me. This standard java class helped me a lot. I used it to reduce all transitions between the server and client down to one line. The server could take the line from the client and know all of its states and requests. It was the simplest way to do it with the restriction of only writing in one file. As a side note I found it very difficult for my mind to wrap this one file style.

The formatting for the JavaOutput.txt file was a little interesting. It wasn’t talked about much in the assignment. But I believe I got it done to the best of my ability and to what the specifications stated.

This was a tricky way to start this class off but I think I did well on it. Everything was met as far as the objectives for the assignment.

WEBSERVER ASSIGNMENT:

This one was a little tricky. Got it working pretty easily by going through the instruction steps very slowly and making sure everything worked at each step of the way.

**NOTE: All information from here forward is taken from the book or from the class lecture/notes. None of it is my original ideas or thoughts or research or words.**

Class one lecture notes:

Class two lecture notes:

Class three lecture notes:

(see notebook!)

***Class four lecture notes:***

Processes:

Process is a program that is being executed. Every process has code being executed.

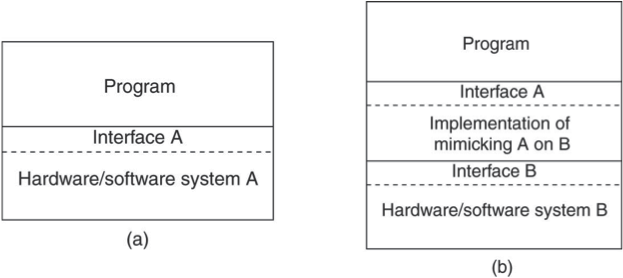
Messages are used to share info between processes. On local machine buffer is used.

Context Switch – when going from process A to process B. each process has information stored about it, like who ran it, how much mem its taking up, saved variables ect. (Process control blocks.) all that is loaded out for A and in for B… 3 steps, 1. Push A out.. 2 let OS do switches.. 3. Push B in. IPC (interprocess communication). Can’t let the processes access low level instructions to manipulate hardware. Needs to go into kernel mode on behalf of the user.

Threads - user level threads -> people can make these to control scheduling where they are managed.

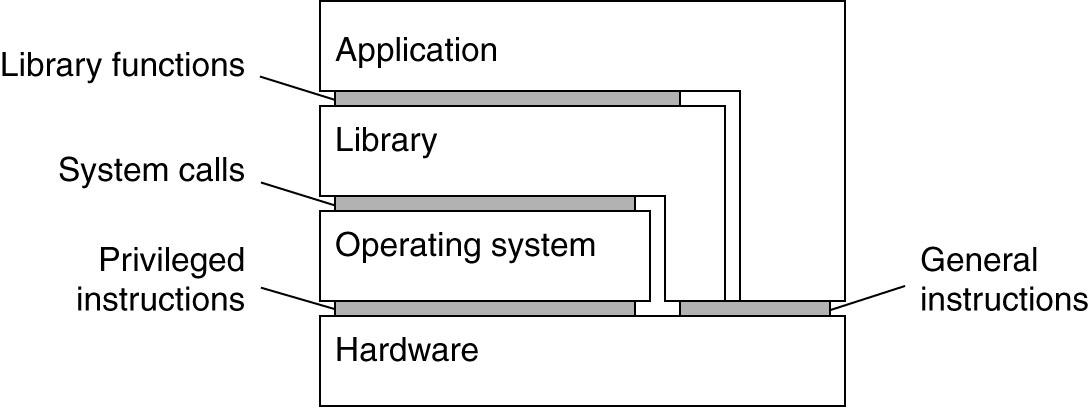
Multithreaded servers: can be implemented differently.

Virtualization in distributed systems:



mimicking hardware and software of system A we can present the same interface to the program.

System calls will be interpreted by the virtualization machine:

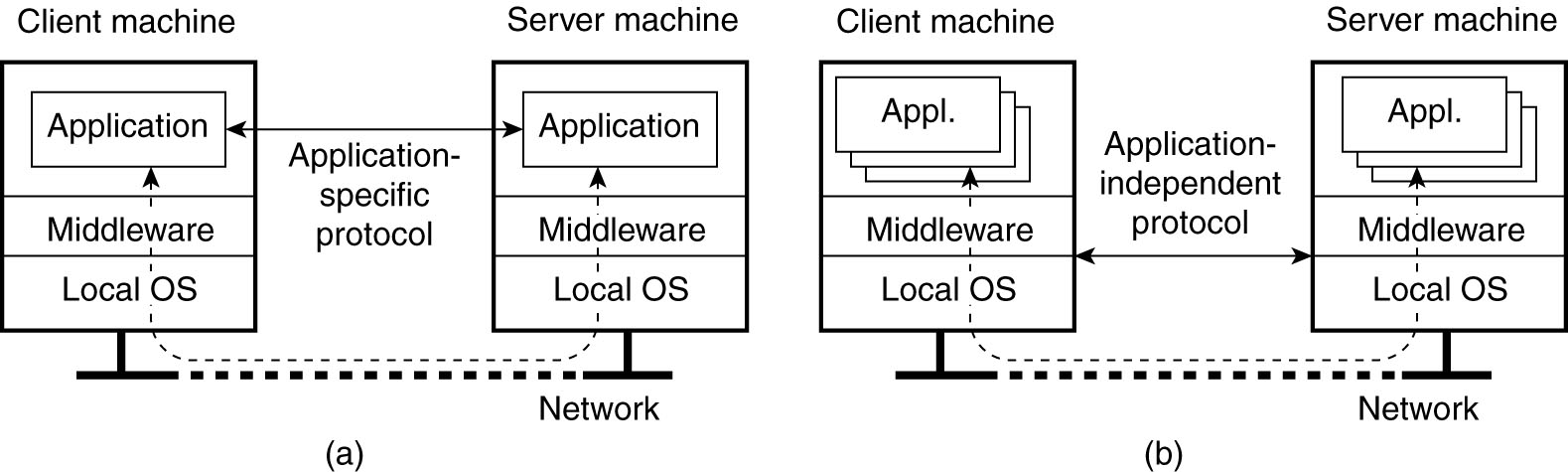


OS can execute general instructions and Privileged Instructions. Application gets to the OS privileged instructions by going through the Library.

Virtual machine monitor: running windows 7 on top of my OS X. presents all the runtime libraries of windows 7.

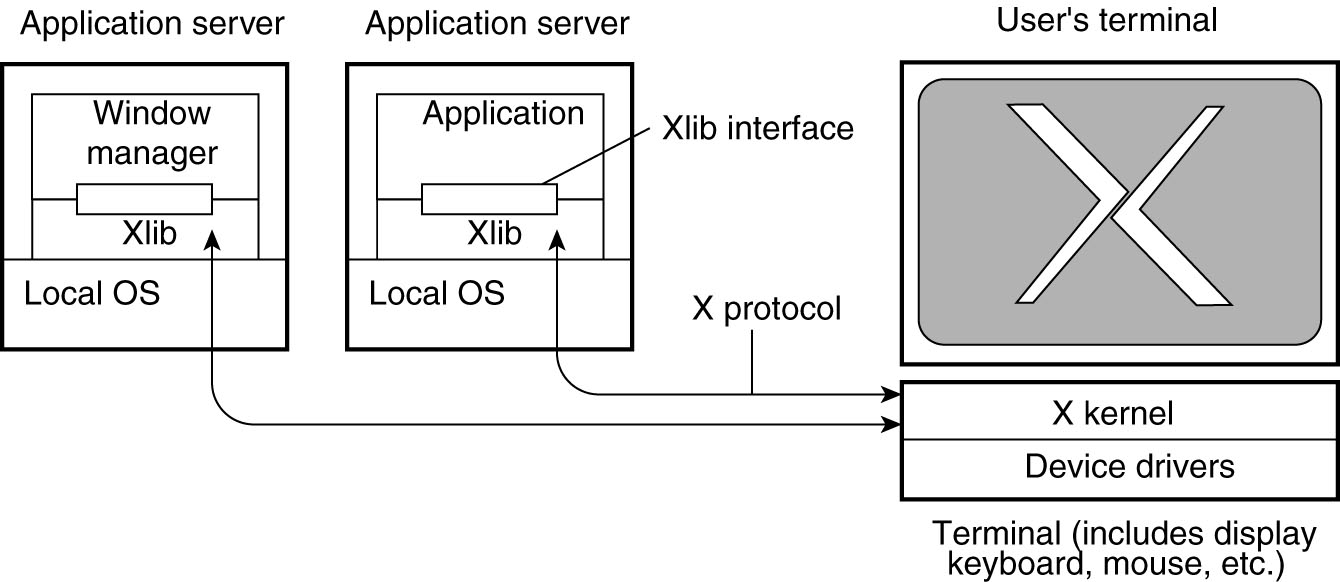
Annecdote about IBM and small shops with dos-vse and mvs-os.

Networked User Interfaces:



Middleware decides the way the OS talks to each other.

XWINDOWS – a system.



The application server is running a DISPLAY server. When it wants something from the application server it sends a request to the display server to show the information. The key is to know that the X kernal user’s client machine is running the display server. Allows for windowing panes.

Clients display applications from application servers.

Design:

Super server model: super server sends clients to other servers

Server clusters: client request get dispatched to nodes that get information in there sector.

Migrating CODE: A client with a code repository

Weak mobility – the code is transferred but then starts from its initial point.

Strong mobility – State of a running program is transferred. It’s running process at some location starts where the program counter is located. (complications on this one from moving to different system types).

To pass a process you need to think about what it is bound to.

•By identifier – Strongest – URL, internet address. The absolute name. Thus the absolute thing.

•By value – such as standard libraries. Copy of the library is fine.

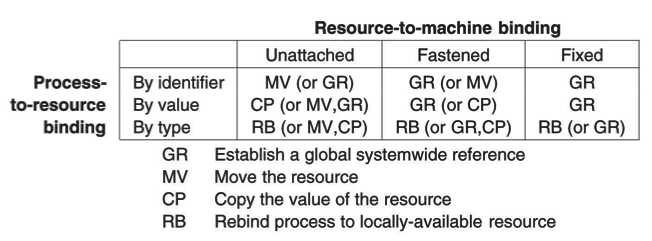
•By type – weakest – reference to “some terminal” or “some printer”

What do you do with that processes resources:

•Unattached resource – such as data file specific to the application (so just move it)

•Fastened resource – could be moved theoretically, but not practical. Complete web site, local database used by others.

•Fixed resource – cannot be moved. E.g., real IP address / end point.



-Actions taken with the respect to the references to local resources when migrating code to another machine.

* talked about myWebServer

***Lecture 5 / midterm review (online view only):***

Midterm review first. General lecture where we left of second.

1st – mutual exclusion problem.

Section of code that can’t be shared.

Difficulty comes with Step N: look – is section open?, N+1: if so set flag to ‘not open’

In between these sections the process may be swapped out. Critical section. We can set these 2 steps to one step. In DS this could be a problem.

Joke server example. – we can store whole state on the client but not on the server.

Thick client vs thin client: very thin client = lots of traffic on network

Thick client = trouble with updating all the machines.

Need a compromise (think adobe flash client with constant update notifications)

ATM networks- asynchronies transmition mode. Achieve better throughput by fixed length packets.

OSI model – layered set of protocols. Run and test at each layer. Write to an interface one layer below what you’re working at.

Typically we don’t use osi, we use tcp/ip. But it’s a useful model for abstraction, and header bits for each level.

Scheme or protocol with (URL) http://\*machine name / ip addr : [port num (usually a default) ] / (information used by the application / looks like directory of text files.)\*

Ex: http://some.machine.gov:1585/textfiles/pdf/manual.html

After the hostname its entirely up to the application protocol to handle what is sent (MyWebServer)

1010111…111 : R1 register. What is it used for? Who knows, depends on the context. Could be big/little endian. Could be float or unsigned int or instruction. Need local architecture. So when shipping bit stream from machine a to machine b, we might have to completely reverse it.

Marshalling = process of moving from one machine to another. (serializing the data) (possible translation to a foreign architecture).

MIME type: the idea is when distributing data form p2p or client to server, need a header to let browser(receiver) know what they’re getting.

Ex: Content-Type: text/html /n/n data

Process communication: p1 and p2 talk through messages. Questions? Do both processes have to be active? No->requires persistence in the messaging system.(store messages somewhere) How do we handle failures? Ect.

Idea of persistent communication: voicemail system. Messages saved if other process isn’t there. Need code to check for messages, and delete checked messages.

Shared memory system: p1 and p2 share memory, their symbol tables look at the same place.(could be going through a network to look for value.)

P2P systems: structural themes, enforcement of good behavior. Bittorrent: must upload to be able to download. To pull recourses out you must give back.

Mobile agents: process that takes its state and moves to different machines. New programming paradigm.

Interesting security concerns: host must protect itself from agent. Agent must protect its own state from host even operating on same filesystem.

Group communication: MULTICAST – humans understand groups well – societies – organizations

Groups are important to security. –admin groups with rights to everything. – can’t give everyone access to everything. Services also grouped into organizations.

Multicast will send out a package for everyone, or specific groups to get. –getting verification for certain things is very important.

Atomic multicast : everyone gets the message or no one does!

-setup a simple local group, pick a number for the group that’s not a conflict. Time to live(TTL) lets the packet hope around a certain amount of times but then doesn’t go to far out of the local system because its TTL is set low.

*GPS*: read up on it. 29 satellites that give global clock.

*Berkley Sockets*: bind and listen. Used under the hood.

*Service Architecture*: server A,B,C,D… all different types. gate keeper directs you where to go and then those server may talk to each other.

Mobile Internet Protocol: idea that you can take you laptop and where and access your joke server. But it will be a different server/port. Need to change dns tables so that the new IP will match up. (moving server is hard because clients won’t know where you’ve gone).

*IP addresses are geographically fixed*!!

They need updating when servers move.

*Virtualization*: each layer presents what looks like actual hardware to the level above, but in actuality it’s just the VM

+Synchronous calls: make the call wait for response.

+Asynchronous calls: make the call and keep working until you get a response. Must decide where to synchronize.(not common to have built in libraries for this, so mostly done on application level)(much more complicated to check if/when data is returned from the call)

XWindows: unusual system. Display server is on the application client. When client makes a request of the application server, then the server goes through the xwindows client and that client talks to the xwindow kernal server on the display server… in essence there is a server on the client and a client on the server,.

Most of the display window is buffered to be displayed.

Distributed HashTables: spread over far areas, resolve local nodes and a process can add or remove itself into the system.

*Distributed computing environment*: register your service as the end point in a table. Same idea as Deamon server(knows where the current location of server and can kill off or start up other servers).

*UDP: universal datagram protocol*: cheap! No resend, no guarantee, no acknowledgement. Don’t need to maintain a connection. By contrast in TCP there is a lot of back and forth, and they’ll arrive, and they’ll arrive in order.

-failures: omissions can occur. Datagrams can arrive out of order.

-cheap because no state stored at source/destination; no extra messages no handshaking.

DHCP: assigning IPs to connecting machines and managing them.

NAT: remapping packets from one IP to another. Data sent to router and the router sends it to the correct machine.

*IDL compiler*: Interface definition language: so precise that it lets us determine through the interface what is going in and out of the computer. (can be done with paper and pencile) DESCRIBES WHAT GOES IN AND WHAT COMES OUT.

RPC(remote procedure calls):1. Client procedure calls client stub in normal way

2. Client stub builds message, calls local OS

3. Client’s OS sends message to remote OS

4. Remote OS gives message to server stub

5. Server stub unpacks parameters, calls server

6. Server does work, returns result to the stub

7. Server stub packs it in message, calls local OS

8. Server’s OS sends message to client’s OS

9. Client’s Os gives message to client stub

10. Stub unpacks result, returns to client

(they pretend to be local procedures from each others prospective)

Architecture: complete specification on paper mapping of the meaning of bits, interpreted as instructions or data for instructions.

Context switch: expensive, switch processes, must go to kernal mode save registers ect… then go to other process.

Running multiple threads is better for this because you don’t need to go through a context switch.

Scale up problem: -administration problem.

JokeServer: asynchronous thread spawn to wait for client and admin client. –what did server state have to do with clients? What information it returned to them.

*Fully distributed algorithms*: they work in the absence of a global clock. If any process fails it can’t make the whole system fail.

*Common gateway interface*: client uses a form. Bundled that data up and parsed and computed, built a file, and then send back

Knowing the right mime type: look at the file extensions. And then associate with the right mime type.

Main drawback of DS: no global clock to number the steps. Time T1 on first machine has no meaning on the second machine. In some cases you can keep track of sequence of events.

Common Theme to all DS: always a compromise! (always a trade off).

How do you marshal a linkedlist across systems? The pointers don’t mean anything on a remote system. It’s something we can’t manage in here. can’t be automatically marshaled.

***Reading notes(aka text book outlined):***

***Chapter1: INTRODUCTION***

Microprocessors spawned the change that created DS.

1.1: Definition of a Distributed System

-A DS is a collection of independent computers that appears to its users as a single coherent system

1.2 GOALS:

-Making resources accessible

-groupware: software for collaborative editing, teleconferencing and so on

-Distributed transparency

-transparent: a DS that appears to a user as a single system.

7 types of transparency:

-*Access*: Hides differences in data representation and how a resource is accessed

-*Location*: Hides where a resource is located.

-*Migration*: Hides that a resource may move to another location.

-*Relocation*: Hides that a resource may be moved to another location while in use.

-*Replication*: Hide that a resource is replicated

-*Concurrency*: Hide that a resource may be shared by several competitive users

-*Failure*: Hide the failure and recovery of a resource.

-Degree of Transparency: Sometimes hiding everything about the system is not a good idea.

-Openness

- A system that offers services according to standard rules that describe the syntax and semantics of those services.

-usually written in interfaces

-IDL: Interface Definition Language.

-Interoperability: characterizes the extent by which two implementations of systems or components from different manufacturers can co-exist .

-Portability: characterizes to what extent an application developed for a DS A can be executed, without modification, on a different DS B that implements the same interfaces as A.

-Extensible: It should be easy to add parts that run on different operating systems.

-Scalability:

-Problems: Centralized services: a single server for all users

-Centralized data: a single on-line telephone book

-Centralized algorithms: Doing routing based on complete information.

~Synchronous connection: When a client is blocked waiting for a response from the server. Very bad for wide DS… not so bad when introduced to LAN DS

~Asynchronous communication: Client will keep running after its request has been sent. Special handler is called when the server returns request.

-Distribution: taking a component and splitting it into smaller parts to carry over the network easier.

DNS: Domain Name System: trees of domains divided into zones. The names in each zone are handled by a single name server.

Replication and Caching help with scalability issues. This is bad for consistency though(old versions of info)

-PITFALLS:

1. The network is reliable

2. The network is secure

3. The network is homogeneous

4. The topology does not change

5. Latency is zero

6. Bandwidth in infinite

7. Transportation cost is zero.

8. There is one administrator.

1.3: TYPES of DISTRIBUTED SYSTEMS:

-Distributed Computing Systems

Cluster Computing: LAN all with same OS. Used for parallel programming in which a single (compute intensive) program is run in parallel on multiple machines.

Grid Computing Systems: LAN without the same OS. Organization is virtual. The people belonging to the same virtual organization have access rights to the resources that are provided to that organization.

OGSA: Open grid services architecture: Complex: various layers and many components.

-Distributed Information Systems

-Distributed transaction: many requests wrapped together and sent to one server than distributed

-EAI: enterprise application integration: letting applications interact with each other.

-Database applications: operations carried out by transactions.

Transactional RPCs(remote procedure calls)

: BEGIN\_TRANSACTION: Mark the start of a transaction

: END\_TRANSACTION: Terminate the transaction and try to commit

: ABORT\_TRANSATION: kill the transaction and restore the old values

: READ: Read data from a file, a table, or otherwise

: WRITE: Write data to a file, a table, or otherwise

ACID - properties of transactions

-Atomic: To the outside world, the transaction happens indivisibly

-Consistent: The transaction does not violate system invariants.

-Isolated: Concurrent transactions do not interfere with each other.

-Durable: Once a transaction commits, the changes are permanent.

Enterprise Application Integration: existed when applications became decoupled from the databases they were built on.

-Distributed Pervasive Systems:

-small, battery-powered, mobile, and having only wireless connection

requirements: 1. Embrace contextual changes

2. Encourage ad hoc composition.

3. Recognize sharing as the default.

Home systems: UPnP: Universal plug and play, standards for universal devices communicating.

Electronic Health Care Systems: gathers information off peoples bodies. Where is it stored? Ect.

Sensor Networks: in-network data processing – used in health care too, computation, storage and transmitting all with the sensors. (big brother?!)

***CHAPTER 2: ARCHITECTURES:***

2.1: ARCHITECTURAL STYLES-

-2 issues with style… the components and the connectors that link those components

1. Layered Architectures:

component at layer L(i) is allowed to call components at L(i-1), but not the other way around. Requests go down while results go up.

2. Object-based architectures: each object corresponds to a component and are connected through remote procedure calls.

3. Data-centered architectures: processes communicate through a common (passive or active) repository

4. Event-based architectures: processes communicate through the propagation of events which optionally carry data. (publish/subscribe system)

2.2: SYSTEM ARCHITECTURES

def: software components, their interaction, and their placement

-Centralized Architectures:

-basic client server: request-reply behavior (synchronous calls)

-idempotent: a request that may be harmful to send more than once(deposit 50k)

-Application layering:

1. The user-interface level: what the user sees

2. the processing level : database queries in and html page listings out

3. the data level: database with web pages (persistent: meaning even if no applications are sending requests the data is still stored somewhere.

-Multitiered architectures:

-two tiered (client/server) divide depends on what u need. Ease of maintaining clients vs load times

--fat clients vs thin clients (UI + app + data on fat ones, just ui on thin ones)

-(physically 3-tiered architecture) 1. Ui 2 application server 3. Database server.

-Decentralized Architectures

-multitiered = vertical distribution. (vertical fragmentation as used in distributed relational databases)

-horizontal distribution: a client or server may physically be split up into logically equivalent parts, but each part is operating on its own share of the complete data set. (p2p systems)

: each client acts as client and server at the same time. (acting as servent)

-overlay network organizes this by representing each process as a linked node

Structured p2p architectures

-organizes system as a distributed hash table(DTH) with membership management.

-Example: Content Addressable Network(CAN): d-dimensional cartesian coordinate space.

Unstructured p2 nodes: rely mostly on randomize algorithms for constructing an overlay network.

-random graph: c neighbors where each neighbor represents a random live node(gives partial view of the entire overlay)

-could bring imbalance by creating popular nodes. (want a semantic overlay network)

Superpeers: part of a CDN (content delivery network)

-maintain and index of their peers and communicate with other superpeers.

-Hybrid Architectures

-edge-server system(provided by ISP) home users connect to the internet through an isp.

-collaborative distributed system: bittorrent—

NOT 2.3-2.4

***CHAPTER 3: PROCESSES:***

3.1 THREADS:

Process table: OS list of processes running on the system.

A process: a program in execution.

Thread context: cpu context plus information for thread management.

*Thread usage in nondistributed systems*:

-context switch: going from process A user space to OS kernel space to process B user space.

--expensive to switch because of saving all memory for process A.

Thread Implementation: often provided in thread packages for applications- creates and destroys.

LWP: lightweight processes- an lwp runs in context of a single process, but multiple LWPs per process.

Scheduler activations: saves management of LWP by kernel. The kernal does an upcall to the thread package.

Threads in Distributed Systems:

* can provide blocking system calls without blocking the entire process.
* Multithreaded servers: dispacther thread reads incoming requests and creates a worker thread to handle those requests.

3.2 VIRTUALIZATION:

Resource virtualization: separation between having a single CPU and being able to pretend there are more can be extended to other resources.

Role of virtualization in DS:

-mimic a foreign OS’s system calls and architecture.

-many different architectures for virtualization.

-process virtual machine: virtualization for just a single processes.

-VMM: virtual machine monitor – runs on top of hardware presenting a virtual hardware for the OS.

3.3 CLIENTS:

Networked User Interfaces: gives a thin-client approach.

Example: The X Window System- control bitmapped terminals.

Heart of the system is formed by what we shall call the X kernel.

-X kernel and X applications need not be on the same machines.

-X protocol is an application level communication by which an instance of Xlib can exchange data and events with the X kernel.

-Window manager given special rights to give the look and feel of the display terminals.

-Thin client: applications for X separate UI and application logic

Modern UIs: - compound document, which can be defined as a collection of documents, possibly of very different kinds(like text, images, spreadsheets, ect.), which are seamlessly integrated at the UI level.

Ideally a client should not be aware its communicating with a remote process.

3.4 SERVERS

General Design Issues:

-iterative server: handles the request itself and may send a response.

-concurrent server: doesn’t handle the request but passes it to a separate thread or another process.

EX: multithreaded server.

-knowing server end points, aka port, that it is listening at is important.

Superserver: listens to many ports and delegates the service at the specific port to a fork() process.

Breaking a server connection:

-user exit the client application.

-can have the server listen at another port for out-of-band data which tells it to end the connection.

Stateless server: does not keep information on connecting clients.

State is important: think jokeServer assignment.

Server Clusters

-three tier organization: 1st client request, 2nd application/compute servers, 3rd distributed file/database

- standard way of accessing a server cluster is to set up a TCP connection over which application level requests can be sent as part of a session.

Distributed Servers: most clusters offer a single access point. When that point fails the cluster becomes unavailable

-We can make several access points with public addresses.

-DNS (domain name system) can return several address all belonging to the same host.

-basic idea of a distributed server: clients benefit from a robust, high-performing, stable server.

CoA- care of address: given to a mobile node when connecting to a foreign network.

-can offer stable addressing.

Managing Server Clusters:

-Server clusters should appear as a single computer.

-admin can manage local control calls to maintain the cluster.

Example: PlanetLab

-server cluster containing many nodes donated from all different organizations

-each node can be broken into a cluster server with process running on vserver on them.

-individual manager nodes to manage the recourses.

-monitoring is don’t with each node having its CPU usage ect easily accessible with HTTP requests

3.5 CODE MIGRATION:

=in DS it is where an entire process is moved from one machine to another.

Approaches:

-mobile agents: small program that moves from site to site looking for something. Such as a search engine request.

Weak mobility: transfer only the code segment, along with perhaps some initialization data. (must start from start of code)

String mobility: execution statement is transferred as well(running process is stopped, moved to a whole new machine and resumes execution where it left off)

-resource to machine binding issues: (see lecture 4 notes above).

Migration in Heterogeneous Systems: before we just assumed all systems with code migrating through were the same.

-highly portable languages: java & scripting languages.

-current practice to deal with heterogeneity is using virtual machines: JVM!

***CHAPTER 4: COMMUNICATION***:

4.1 FUNADAMENTALS

Layered protocols – based on lack of shared memory all communication in DS is based on sending and receiving low level messages.

ISO(International Standards Organization) – reference model that clearly defines various levels involved, and points out which level should do which job.

-the protocols from this model were never widely used, but the model is useful for understanding networks.

Designed to allow open systems to communicate with connection oriented protocols.

7 protocol layers, each with its own header bits to the message, each level has an interface to interact with the level below it. A collection of protocols is often called a protocol stack.

*Lower level protocols*:

-physical layer is transmitting 0s and 1s, how many bits per second can be sent and whether transmission can take place in both directions

- data link layer: detects and corrects errors in physical layer. Puts bits into groups called frames to check.

-frames are assigned sequence numbers in the header to tell them apart.

-network layer: concerned mostly with routing (jumping from network to network until it reaches the destination)

-complicated by the shortest route not always being the best.

-IP(internet protocol) is most widely use network protocol.

*Transport protocols*:

Transport layer: implements all the services that are not provided at the interface of the network layer.

TCP/IP: transmission control protocol

Higher-level protocols:

OSI has 3 layers above the transport protocol. The application layer is only used in practice.

Session layer: enhanced version of the transport layer: provides dialogue control.

Presentation layer: gets meaning out of the bits sent and received at lower levels.

Application layer: HTTP and FTP SMTP, ect ect

Middleware protocols:

Lives in the application level. Contains many general-purpose protocols that warrant their own layers.

Types of communication:

-asynchronous communication= make a remote call and then go about your business, possibly get interrupted later to handle return from call.

-synchronous communication= make a remote call and sit waiting for response.

4.2 REMOTE PROCEDURE CALL (RPC)

- when machine A makes a call to Machine B, process A is suspended and execution of the called procedure takes place on B. much like a function call, while no message passing at all is visible to the programmer.

-different address spaces cause subtle complications

Basic RPC operation.

-call-by-reference: a pointer is passed to a function (bad when not sharing same address space)

-cal by value: a value is passed to a function

=main idea of RPC is to make a remote call look as close as it possibly can to a local one.

See RPC discussion in lecture 5 notes!

Parameter Marshaling: packing parameters into a message.

Passing reference parameters: could 1. Forbid pointers. 2. Pass what is being pointed at, if its output just need to send it back, if it’s just input you can pass it in but not pass it back.

4.3 MESSAGE ORIENTED COMMUNICATION:

Berkeley sockets: aka sockets interface.

-conceptually a communication end point to which an application can write data that are to be sent out over the overlay network.

Message-Passing Interface (MPI) – standard for message passing. Communication takes place within a known group of processes. Each process in the group is also assigned a local identifier.

Message-queuing systems: Message Oriented middleware(MOM): provide extensive support for persistent asynchronous communication.

* queue managers: handle the queuing layer of messages.(long been around in email services.

4.4 STREAM-ORIENTED COMMUNICATION

- support for continuous media.

- simple stream: consists of only a single sequence of data.

- complex stream: consists of several related simple streams (substreams)

Quality of Service: timing and other nonfunctional requirements.

* Enforcing QoS : differentiated services- means for differentiating classes of data.

Stream Synchronization:

* effect of packet loss in interweaving transmission. (not too bad for video if a lost packet, bad for video if lost packet occurs in noninterleaved transmission)
* MPEG(motion picture experts group) standards form a collection of algorithms for compressing video and audio.
* MPEG-2 for example compresses broadcast video to 4-6mbps.

4.5 MULTICAST COMMUNICATION

(see notes above from class 5)

* sending data from one to multiple receivers
* can be divided into groups.
* Gossip based data dissemination: epidemic protocol: rapidly propagate information among a large collection of nodes using only local information.