CS865 – Distributed Software Development
Lecture 3

Tannenbaum and Van Steen – Chapter 3

Processes

Introduction to Threads

Processes (Operating Systems Tutorial)
- An operating system creates a number of virtual processors, each one for running a different program.
- The operating system has a process table to keep track of these virtual processors.
- The process table contains entries to store CPU register values, memory maps, open files, accounting information, privileges, etc.
- A process is a running instance of a program, including all variables and other state attributes on one of the operating system's virtual processors.

- The operating system ensures that independent processes cannot affect each other's behavior.
- Sharing the same CPU and other hardware resources is made transparent with hardware support to enforce this separation.

- Each time a process is created, the operating system must create a complete independent address space.
  e.g., zeroing a data segment, copying the associated program into a text segment, and setting up a stack for temporary data.
- Switching the CPU between two processes requires:
  - saving the CPU context (which consists of register values, program counter, stack pointer, etc.),
  - modifying registers of the memory management unit (MMU)
  - invalidate address translation caches such as in the translation lookaside buffer (TLB) a cache in a CPU that is used to improve the speed of virtual address translation.
- If the operating system supports more processes than it can simultaneously hold in main memory, it may have to swap processes between main memory and disk before the actual switch can take place.

Threads (C ProgrammingTutorial and Thread Chapter)
- A traditional process has a single thread of control and a single program counter.
- Modern operating systems provide multiple threads of control within a process at the kernel level - these are called Threads or lightweight processes. (A heavyweight process is an instance of a program).
- All threads within a process share the same address space.
- A thread shares its code and data section with other threads.
  - Each thread has its own program counter, stack and register set.
    - The *program counter* determines which instruction the thread is currently executing.
    - The *register set* saves the thread's state when it is suspended and reloaded into the machine registers upon resumption.

Thread Types
User Threads
- Threads are implemented at the user level by a thread library
  - Library provides support for thread creation, scheduling and
  - management.
  - User threads are fast to create and manage.

Kernel Threads
- Supported and managed directly by the OS.
  - Thread creation, scheduling and management take place in kernel space.
  - Slower to create and manage.

Thread Usage in Nondistributed Systems

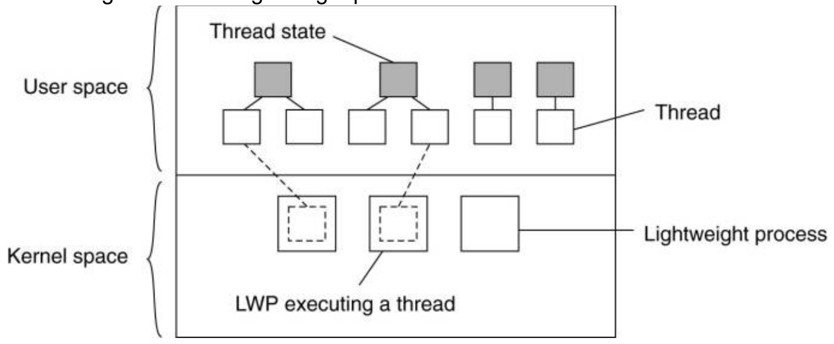
Benefits of multithreaded processes:
- Increased responsiveness to user:
  - A program continues running with other threads even if part of it is blocked or performing a lengthy operation in one thread.
- Resource Sharing
  - Threads share memory and resources of their process.
- Economy
  - Less time consuming to create and manage threads than processes as threads share resources, e.g., thread creating is 30 times faster than process creating in Solaris.
- Utilization of Multiprocessor Architectures
  - Increases concurrency because each thread can run in parallel on a different processor.
- Many applications are easier to structure as a collection of cooperating threads.
  - e.g., word processor - separate threads can be used for handling user input, spelling and grammar checking, document layout, index generation, etc.

Thread Implementation

- Threads are provided in the form of a thread package.
- The package contains operations to create and destroy threads as well as operations on synchronization variables such as mutexes and condition variables.
- Two approaches to implement a thread package.
  1. Construct a thread library that is executed entirely in user mode.
     Advantages:
     - it is cheap to create and destroy threads
       - all thread administration is kept in the user's address space, the price of creating a thread is primarily determined by the cost for allocating memory to set up a thread stack
       - destroying a thread mainly involves freeing memory for the stack, which is no longer used.
     - switching thread context can be done in just a few instructions
     Disadvantage:
     - A blocking system call will immediately block the entire process to which the thread belongs, and thus also all the other threads in that process
  2. Have the kernel be aware of threads and schedule them.
     Advantages
     - Eliminates blocking problem.
     Disadvantage:
     - Every thread operation (creation, deletion, synchronization, etc.), will have to be carried out by the kernel, requiring a system call.

     Solution: hybrid of user-level and kernel-level threads, generally referred to as lightweight processes (LWP).
     - An LWP runs in the context of a single (heavyweight) process
     - There can be several LWPs per process.
     - User-level thread packages can be used as well.
       - The package provides facilities for thread synchronization, such as mutexes and condition variables.
         - Mutex - ***mu**tual **ex**clusion object*.
           - A mutex is a program object that allows multiple program threads to share the same resource, such as file access, but not simultaneously.
           - When a program is started, a mutex is created with a unique name.
           - After this stage, any thread that needs the resource must lock the mutex from other threads while it is using the resource.
           - The mutex is set to unlock when the data is no longer needed or the routine is finished.
       - The thread package is implemented entirely in user space in which all operations on threads are carried out without intervention of the kernel.
       - The thread package can be shared by multiple LWPs
         - each LWP can run its own (user-level) thread.
         - Multithreaded applications are constructed by creating threads, and subsequently assigning each thread to an LWP
         - Assigning a thread to an LWP is normally implicit and hidden from the programmer.

Combining kernel-level lightweight processes and user-level threads.



Multithreading Models

Three common ways of establishing a relationship between user level threads and kernel-level threads

1. Many-to-One
   Many user-level threads mapped to single kernel thread.
   - Easier thread management.
   - Blocking-problem.
   - No concurrency.
   - Examples: Green threads for Solaris

2. One-to-One

   Each user-level thread maps to a kernel thread.
   - Overhead of creating kernel threads, one for each user thread.
   - No blocking problem
   - Provides concurrency.
   - Examples: Linux, family of Windows

3. Many-to-Many

   Allows many user level threads to be mapped to many kernel threads.
   - Allows the OS to create a sufficient number of kernel threads.
   - Users can create as many as user threads as necessary.
   - No blocking and concurrency problems.
   - Two-level model.

Threading Issues

fork and exec (two basic tread operations)

- Change in semantics of **fork()** and **exec()** system calls.
- Two versions of **fork** system call:
  - One duplicates only the thread that invokes the call.
  - Another duplicates all the threads, i.e., duplicates an entire process.
- **exec** system call:
  - Program specified in the parameters to **exec** will replace the entire process – including all threads.
- If **exec** is called immediately after forking, duplicating all threads is not required.

Cancellation

- Task of terminating a thread before it has completed.
  - Canceling one thread in a multithreaded searching through a database.
  - Stopping a web page from loading.
- Asynchronous cancellation
  - One thread immediately terminates the target thread (one to be canceled).
- Deferred cancellation
  - The target thread can periodically check if it should terminate, allowing a normal exit.

Signal Handling

- Signal to notify a process that a particular event has occurred.
  - Default or user defined signal handler.
- Synchronous signal is related to the operation performed by a running process.
  - Illegal memory access or division by zero.
- Asynchronous signal is caused by an event external to a running process.
  - Terminating a process (<control><C>) or a timer expires.
- Options for delivering signals in a multithreaded process:
  - Signal to the thread to which the signal applies.
  - Signal to all threads.
  - Signal to certain threads.
  - Signal to a specific thread.

Thread Pools

- Create a number of threads at process startup and place them into a pool where they sit and wait for work.
  - e.g. for multithreading a web server.
- A thread from the pool is activated on the request, and it returns to the pool on completion.
- Benefits of thread pool:
  - Faster service
  - Limitation on the number of threads, according to the need.
- Thread-pool-architecture allows dynamic adjustment of pool size.

Specific Data

- Certain data required by a thread. These data are not shared with the other threads.
  - Example of thread-specific data:
- In a transaction processing system, different transaction services will be provided by different threads.

Scheduler Activations

- Communication between the user-thread library and the kernel threads.
- An intermediate data structure - LWP.
  - User thread runs on a virtual processor (LWP)
  - Corresponding kernel thread runs on a physical processor
- Each application gets a set of virtual processors from OS

- Application schedules threads on these processors
- Kernel informs an application about certain events issuing upcalls which are handled by thread library.

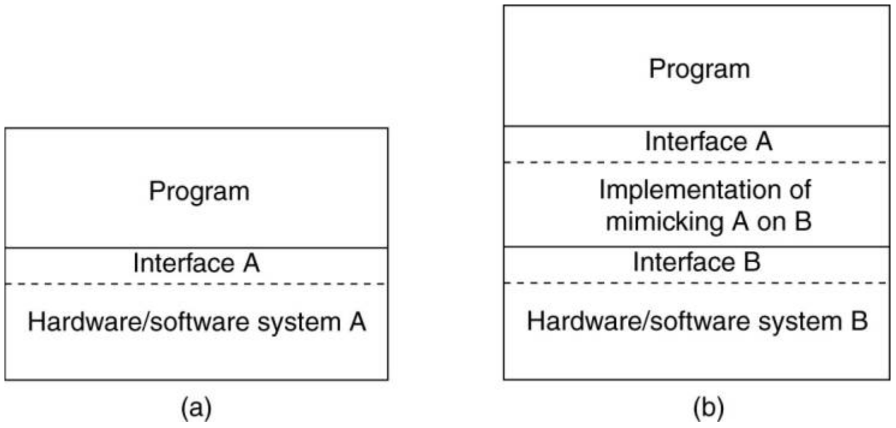
Advantages of LWPs in combination with a user-level thread package:

1. Creating, destroying, and synchronizing threads is relatively cheap and involves no kernel intervention at all.
2. Provided that a process has enough LWPs, a blocking system call will not suspend the entire process.
3. There is no need for an application to know about the LWPs. All it sees are user-level threads.
4. LWPs can be easily used in multiprocessing environments, by executing different LWPs on different CPUs.

Virtualization (see e.g. wikipedia)

- **Virtualization** is a broad term that refers to the abstraction of computer resources.
- Virtualization creates an external interface that hides an underlying implementation

(a) General organization between a program, interface, and system.
(b) General organization of virtualizing system A on top of system B.


Program
Interface A
Hardware/software system A
(a)
Program
Interface A
Implementation of mimicking A on B
Interface B
Hardware/software system B
(b)

- Common uses of the term, divided into two main categories:

1. **Platform virtualization** involves the simulation of virtual machines.
   - Platform virtualization is performed on a given hardware platform by "host" software (a *control program*), which creates a simulated computer environment (a *virtual machine*) for its "guest" software.
   - The "guest" software, which is often itself a complete operating system, runs just as if it were installed on a stand-alone hardware platform.

2. **Resource virtualization** involves the simulation of combined, fragmented, or simplified resources.
- Virtualization of specific system resources, such as storage volumes, name spaces, and network resources.

Role of Virtualization in Distributed Systems

Issue:
- While hardware and low-level systems software change reasonably fast, software at higher levels of abstraction (e.g., middleware and applications), are much more stable - legacy software cannot be maintained in the same pace as the platforms it relies on.

Solution:
- Virtualization can help here by porting the legacy interfaces to the new platforms and thus immediately opening up the latter for large classes of existing programs.

Issue:
- Networking has become completely pervasive.
- Connectivity requires that system administrators maintain a large and heterogeneous collection of server computers, each one running very different applications, which can be accessed by clients.

Solution:
- The diversity of platforms and machines can be reduced by letting each application run on its own virtual machine, possibly including the related libraries and operating system, which, in turn, run on a common platform.

Issue:
- Management of content delivery networks that support replication of dynamic content becomes easier if edge servers supported virtualization, allowing a complete site, including its environment to be dynamically copied.
- As we will discuss later, it is primarily such portability arguments that

Solution:

- Virtualization provides a high degree of portability and flexibility making it an important mechanism for distributed systems.

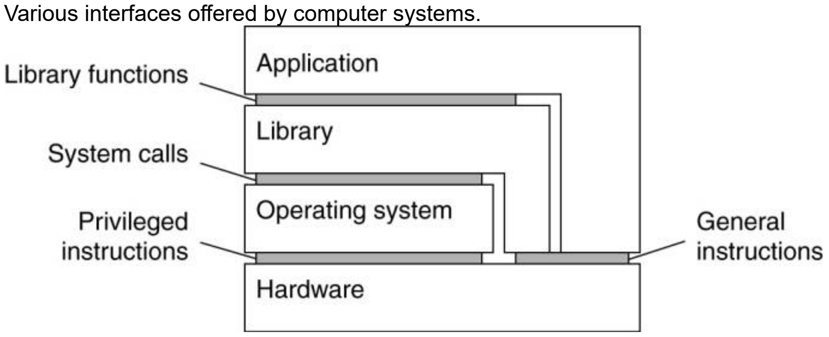
Architectures of Virtual Machines

Variety of ways to realize virtualization. (overview in (Smith and Nair 2005))

Four distinct levels of interfaces to computers the behavior of which that virtualization can mimic:

1. An interface between the hardware and software, consisting of machine instructions that can be invoked by any program.
2. An interface between the hardware and software, consisting of machine instructions that can be invoked only by privileged programs, such as an operating system.
3. An interface consisting of system calls as offered by an operating system.
4. An interface consisting of library calls, generally forming what is known as an application programming interface (API). In many cases, the aforementioned system calls are hidden by an API.

Various interfaces offered by computer systems.

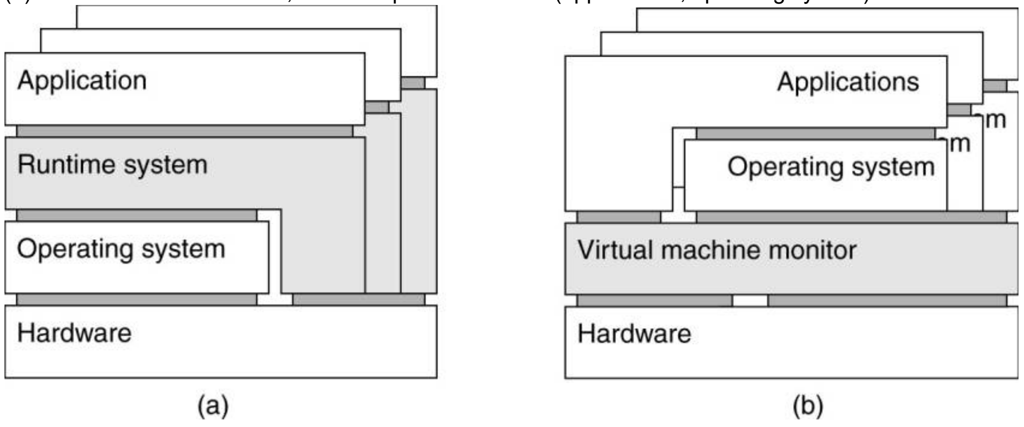


Virtualization can take place in two different ways:

1. Process virtual machine
   - Build a runtime system that provides an abstract instruction set that is to be used for executing applications.
   - Instructions can be interpreted (as is the case for the Java runtime environment), but could also be emulated as is done for running Windows applications on UNIX platforms.

2. Virtual machine monitor (VMM)
   - Implemented as a layer completely shielding the original hardware, but offering the complete instruction set of that same (or other hardware) as an interface.
   - This interface can be offered simultaneously to different programs.
   - Possible to have multiple, and different operating systems run independently and concurrently on the same platform.
   - Example: VMware (Sugerman et al., 2001) and Xen (website) (Barham et al., 2003).

(a) A process virtual machine, with multiple instances of (application, runtime) combinations.
(b) A virtual machine monitor, with multiple instances of (applications, operating system) combinations.



- VMMs will become increasingly important in the context of reliability and security for (distributed) systems. (Rosenblum and Garfinkel 2005)
    - Since they allow for the isolation of a complete application and its environment, a failure caused by an error or security attack need no longer affect a complete machine.
    - Portability is greatly improved as VMMs provide a further decoupling between hardware and software, allowing a complete environment to be moved from one machine to another.

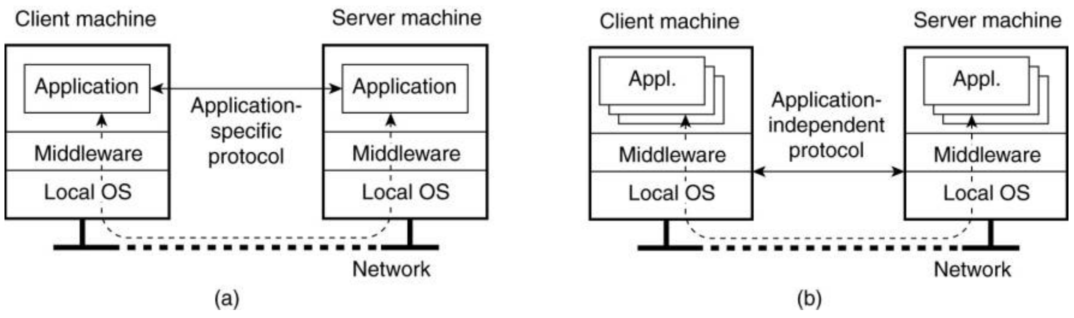
Clients

Networked User Interfaces

Two ways to support client-server interaction:
1. For each remote service - the client machine will have a separate counterpart that can contact the service over the network.
   Example: an agenda running on a user's PDA that needs to synchronize with a remote, possibly shared agenda.
   In this case, an application-level protocol will handle the synchronization, as shown in Figure(a).

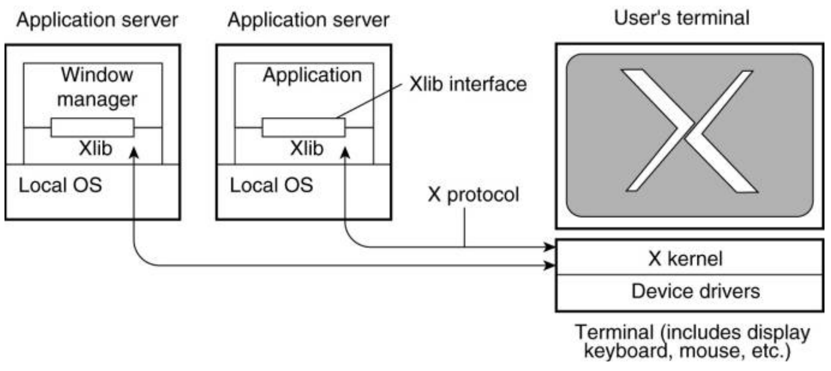
2. Provide direct access to remote services by only offering a convenient user interface.
   The client machine is used only as a terminal with no need for local storage, leading to an application neutral solution as shown in Figure(b).
    - - thin-client approach - everything is processed and stored at the server

(a) A networked application with its own protocol. (b) A general solution to allow access to remote applications.



Example: The X Window System (X)

- Used to control bit-mapped terminals, which include a monitor, keyboard, and a pointing device such as a mouse.
- Viewed as that part of an operating system that controls the terminal.
- X kernel is heart of the system.
    - Contains all the terminal-specific device drivers - highly hardware dependent.
    - X kernel offers a low-level interface for controlling the screen and for capturing events from the keyboard and mouse.
    - This interface is made available to applications as a library called Xlib.
- The basic organization of the X Window System.



- X kernel and the X applications need not necessarily reside on the same machine.
- X provides the X protocol, which is an application-level communication protocol by which an instance of Xlib can exchange data and events with the X kernel.
  Example:

  - Xlib can send requests to the X kernel for creating or killing a window, setting colors, and defining the type of cursor to display, among many other requests.
  - The X kernel will react to local events such as keyboard and mouse input by sending event packets back to Xlib.

- Several applications can communicate at the same time with the X kernel.
- One specific application that is given special rights - the window manager (WM).
  - WM can dictate the "look and feel" of the display as it appears to the user.
    - The window manager can prescribe how each window is decorated with extra buttons, how windows are to be placed on the display, and so.
    - Other applications will have to adhere to these rules.

How the X window system fits into clientserver computing:
- The X kernel receives requests to manipulate the display.
- The X kernel acts as a server, while the applications play the role of clients.
- This terminology has been adopted by X, and although strictly speaking is correct, it can easily lead to confusion.

Thin-Client Network Computing

- Applications manipulate a display using the specific display commands as offered by X.
- These commands are sent over the network where they are executed by the X kernel on the server.

Issue:
- Applications written for X should separate application logic from user-interface commands
- This is often not the case - much of the application logic and user interaction are tightly coupled, meaning that an application will send many requests to the X kernel for which it will expect a response before being able to make a next step. (Lai and Nieh 2002)
- Synchronous behavior adverselys affects performance when operating over a wide-area network with long latencies.

Solutions:
1. Re-engineer the implementation of the X protocol, as is done with NX (Pinzari, 2003).
   i. Concentrates on bandwidth reduction by compressing X messages.
   ii. Overall, this technique has reported bandwidth reductions up to a factor 1000, which allows X to also run through low-bandwidth links of only 9600 kbps.
   iii. X still requires having a display server running.
      - May be asking a lot, especially if the display is something as simple as a cell phone.
      - One solution to keeping the software at the display very simple is to let all the processing take place at the application side.
        - This means that the entire display is controlled up to the pixel level at the application side.
        - Changes in the bitmap are then sent over the network to the display, where they are immediately transferred to the local frame buffer.
      - Approach requires sophisticated compression techniques in order to prevent bandwidth availability to become a problem.
        - Example: a video stream at a rate of 30 frames per second on a 320 x 240 screen
          - If each pixel is encoded by 24 bits - without compression we would need a bandwidth of approximately - 53 Mbps.
          - Compression is needed
          - Compression requires decompression at the receiver - computationally expensive without hardware support.
          - Hardware support can be provided, but this raises the devices cost.

Issue:
Major drawback of sending raw pixel data in comparison to higher-level protocols such as X is that it is impossible to make any use of application semantics.

Solution:
THINC - a few high-level display commands that operate at the level of the video device drivers. (Baratto et al. 2005)
- Commands are device dependent, more powerful than raw pixel operations, but less powerful than X.
- The result is that display servers can be much simpler, which is good for CPU usage, while at the same time application-dependent optimizations can be used to reduce bandwidth and synchronization.

- In THINC, display requests from the application are intercepted and translated into the lower level commands.
- By intercepting application requests, THINC can make use of application semantics to decide what combination of lower level commands can be used best.
- Translated commands are not immediately sent out to the display, but are instead queued.
  - By batching several commands it is possible to aggregate display commands into a single one, leading to fewer messages.
- Instead of letting the display ask for refreshments, THINC always pushes updates as they come available.
  - This push approach saves latency as there is no need for an update request to be sent out bythe display.

Compound Documents

- Modern user interfaces allow applications to share a single graphical window, and to use that window to exchange data through user actions.
- Key idea behind these user interfaces is the notion of a compound document
    - a collection of documents (text, images, spreadsheets, etc.), which are seamlessly integrated at the user-interface level.
- A user interface that can handle compound documents hides the fact that different applications operate on different parts of the document.
- Applications associated with a compound document do not have to execute on the client's machine.
- However, user interfaces that support compound documents may have to do more processing than those that do not.

Client-Side Software for Distribution Transparency

Distribution transparency

- A client should not be aware that it is communicating with remote processes.

Access transparency

- Handled through the generation of a client stub from an interface definition of what the server has to offer.
- The stub provides the same interface as available at the server, but hides the possible differences in machine architectures, as well as the actual communication.

Location, Migration, and Relocation transparency:

- A naming system is crucial
- Cooperation with client-side software is important
    - Example:
        - when a client is already bound to a server, the client can be directly informed when the server changes location.
        - the client's middleware can hide the server's current geographical location from the user, and also transparently rebind to the server if necessary.

Replication transparency

- Implemented as client-side solutions.
    - Example: a distributed system with replicated servers
        - Replication can be achieved by forwarding a request to each replica (figure).
        - Client-side software can transparently collect all responses and pass a single response to the client application.
        - Transparent replication of a server using a client-side solution.

Failure transparency.

- Masking communication failures with a server is typically done through client middleware.
    - Example: client middle-ware can be configured to repeatedly attempt to connect to a server
    - Client middleware can return data it had cached during a previous session, as is sometimes done by Web browsers that fail to connect to a server.

Concurrency transparency

- Handled through special intermediate servers, notably transaction monitors, and requires less support from client software.

Persistence transparency

- Completely handled at the server.

Servers

General Design Issues

- A server is a process implementing a specific service on behalf of a collection of clients.

- Each server is organized in the same way:
  - it waits for an incoming request from a client
  - ensures that the request is fulfilled
  - it waits for the next incoming request.

Several ways to organize servers:

Iterative server:
Iterative server handles request, then returns results to the client; any new client requests *must wait* for previous request to complete (also useful to think of this type of server as *sequential*).

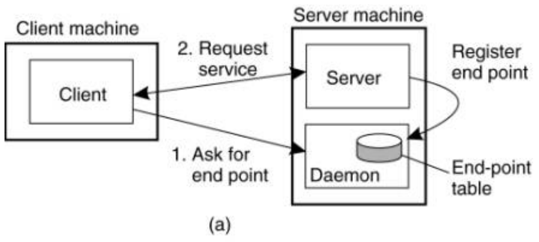
Concurrent server
Concurrent: server does not handle the request itself; a separate thread or sub-process handles the request and returns any results to the client; the server is then free to immediately service the next client (i.e., there's no waiting, as service requests are processed in *parallel*).
- A multithreaded server is an example of a concurrent server.
- An alternative implementation of a concurrent server is to fork a new process for each new incoming request.
  - This approach is followed in many UNIX systems.
- The thread or process that handles the request is responsible for returning a response to the requesting client.

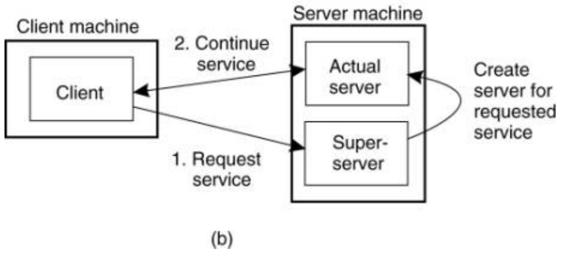
Where do clients contact a server?
- Clients send requests to an end point, also called a port, at the machine where the server is running.
- Each server listens to a specific end point.

How do clients know the end point of a service?
1. Globally assign end points for well-known services.
   - Examples:
     1. servers that handle Internet FTP requests always listen to TCP port 21.
     2. an HTTP server for the World Wide Web will always listen to TCP port 80.
   - These end points have been assigned by the Internet Assigned Numbers Authority (IANA).
   - With assigned end points, the client only needs to find the network address of the machine where the server is running.

- Many services that do not require a pre-assigned end point.
  - Example:
    A time-of-day server may use an end point that is dynamically assigned to it by its local operating system.
    - A client will look need to up the end point.
    - Solution:
      - a daemon running on each machine that runs servers.
      - The daemon keeps track of the current end point of each service implemented by a co-located server.
      - The daemon itself listens to a well-known end point.
      - A client will first contact the daemon, request the end point, and then contact the specific server (Figure(a))

(a) Client-to-server binding using a daemon. (b) Client-to-server binding using a superserver.





- Common to associate an end point with a specific service.
  - Implementing each service by means of a separate server may be a waste of resources.
    - Example: UNIX system
      - many servers run simultaneously, with most of them passively waiting for a client request.
      - Instead of having to keep track of so many passive processes, it is often more efficient to have a single superserver listening to each end point associated with a specific service, (Figure (b))
      - This is the approach taken, with the inetd daemon in UNIX.
        - Inetd manages Internet services by listening to a number of well-known ports for these services.
        - When a request comes in, the daemon forks a process to take further care of the request.
        - That process will exit after it is finished.

Design issue
State of server:
A **stateless server** is a server that treats each request as an independent transaction that is unrelated to any previous request.

- A stateless server does not keep information on the state of its clients, and can change its own state without having to inform any client.
    - Example:
      A Web server is stateless.
        - It merely responds to incoming HTTP requests, which can be either for uploading a file to the server or (most often) for fetching a file.
        - When the request has been processed, the Web server forgets the client completely.
        - The collection of files that a Web server manages (possibly in cooperation with a file server), can be changed without clients having to be informed.

- Form of a stateless design - soft state.
    - The server promises to maintain state on behalf of the client, but only for a limited time.
    - After that time has expired, the server falls back to default behavior, thereby discarding any information it kept on account of the associated client.
    - Soft-state approaches originate from protocol design in computer networks, but can be applied to server design (Clark, 1989; and Lui et al., 2004).

A **stateful server** remembers client data (state) from one request to the next.

- Information needs to be explicitly deleted by the server.
- Example:
    - A file server that allows a client to keep a local copy of a file, even for performing update operations.
    - The server maintains a table containing (client, file) entries.
    - This table allows the server to keep track of which client currently has the update permissions on which file and the most recent version of that file.
- Improves performance of read and write operations as perceived by the client.

Advantages / Disadvantages

- Using a stateless file server, the client must specify complete file names in each request specify location for reading or writing re-authenticate for each request
- Using a stateful file server, the client can send less data with each request
- A stateful server is simpler
- A stateless server is more robust: lost connections can't leave a file in an invalid state rebooting the server does not lose state information rebooting the client does not confuse a stateless server

Distinction between (temporary) session state and permanent state. (Ling et al. 2004)

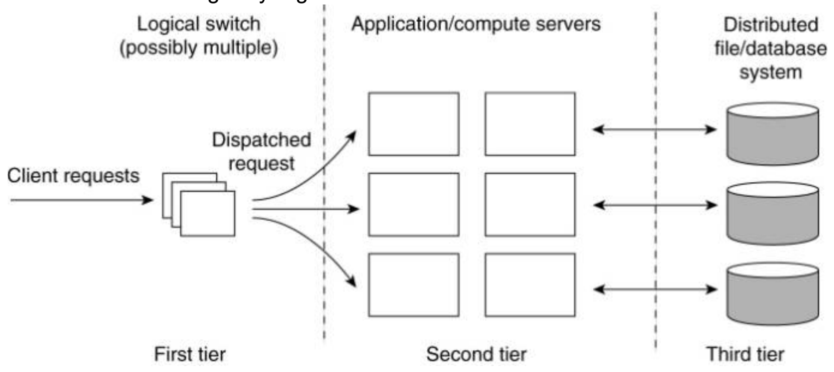
- Session state is maintained in three-tiered client-server architectures, where the application server needs to access a database server through a series of queries before being able to respond to the requesting client.
- No real harm is done if session state is lost, provided that the client can simply reissue the original request.
- Permanent state information is maintained in databases, such as customer information, keys associated with purchased software, etc.
- For most distributed systems, maintaining session state already implies a stateful design requiring special measures when failures do happen and making explicit assumptions about the durability of state stored at the server.

Server Clusters

General Organization

A server cluster is a collection of machines connected through a network, where each machine runs one or more servers.

A server cluster is logically organized into three tiers


Logical switch (possibly multiple)
Application/compute servers
Distributed file/database system
Dispatched request
Client requests
First tier
Second tier
Third tier

First tier - consists of a (logical) switch through which client requests are routed.
    Switches vary:

- Transport-layer switches accept incoming TCP connection requests and pass requests on to one of servers in the cluster,

- A Web server that accepts incoming HTTP requests, but that partly passes requests to application servers for further processing only to later collect results and return an HTTP response.

Second tier: application processing.
Cluster computing servers run on high-performance hardware dedicated to delivering compute power.
Enterprise server clusters - applications may need to run on relatively low-end machines, as the required compute power is not the bottleneck, but access to storage is.

Third tier: data-processing servers - notably file and database servers.
- These servers may be running on specialized machines, configured for high-speed disk access and having large server-side data caches.

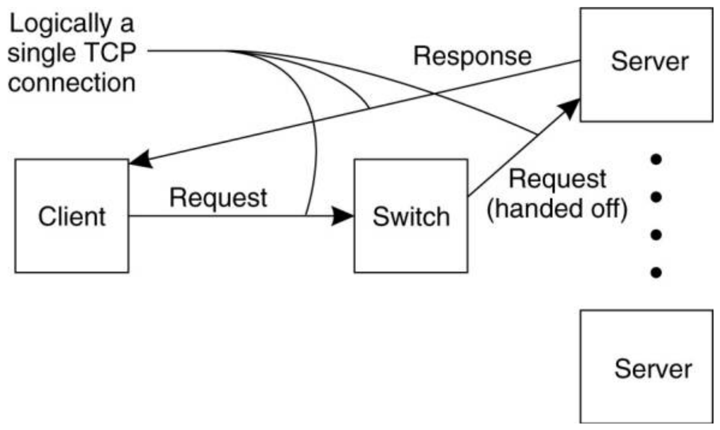
Issue:
When a server cluster offers multiple different machines may run different application servers.
- The switch will have to be able to distinguish services or otherwise it cannot forward requests to the proper machines.
- Many second-tier machines run only a single application.
- This limitation comes from dependencies on available software and hardware, but also that different applications are often managed by different administrators.
- Consequence - certain machines are temporarily idle, while others are receiving an overload of requests.

Solution:
Temporarily migrate services to idle machines to balance load.
Use virtual machines allowing a relative easy migration of code to real machines. (Awadallah and Rosenblum 2004)

The Switch
Design goal for server clusters - access transparency - client applications running on remote machines should not know the internal organization of the cluster.

Implementation - a single access point employing a dedicated machine.
The switch forms the entry point for the server cluster, offering a single network address.
(Note: for scalability and availability, a server cluster may have multiple access points, where each access point is then realized by a separate dedicated machine)

- Standard way of accessing a server cluster - a TCP connection over which application-level requests are sent as part of a session.
  - A session ends by tearing down the connection.
- The principle of TCP handoff - in the case of transport-layer switches, the switch accepts incoming TCP connection requests, and hands off such connections to one of the servers (Hunt et al, 1997; and Pai et al., 1998).
  - When the switch receives a TCP connection request, it identifies the best server for handling that request, and forwards the request packet to that server.
  - The server, will send an acknowledgment back to the requesting client, but inserting the switch's IP address as the source field of the header of the IP packet carrying the TCP segment.
    - Note that this spoofing (the creation of TCP/IP packets using another IP address) is necessary for the client to continue executing the TCP protocol: it is expecting an answer back from the switch, not from some arbitrary server it is has never heard of before.



The Switch is performing load balancing by deciding which server can handle further processing of the request.
- Simplest load-balancing - round robin: each time it picks the next server from its list to forward a request to.

Distributed Servers

Stability vs. Flexibility

Stability –

- Advantage -long-living static access point is a desirable feature from a client's and a server's perspective
- Disadvantage - when that point fails, the cluster becomes unavailable

Flexibility -

- Advantage – dynamically reconfigured cluster when needed.
- Disadvantage – more administrative overhead

Example: a design of a distributed server from a dynamically changing set of end-user machines, with also possibly varying access points, which appears to the outside world as a single, powerful machine. (Szymaniak et al. 2005).

Creating a stable access point

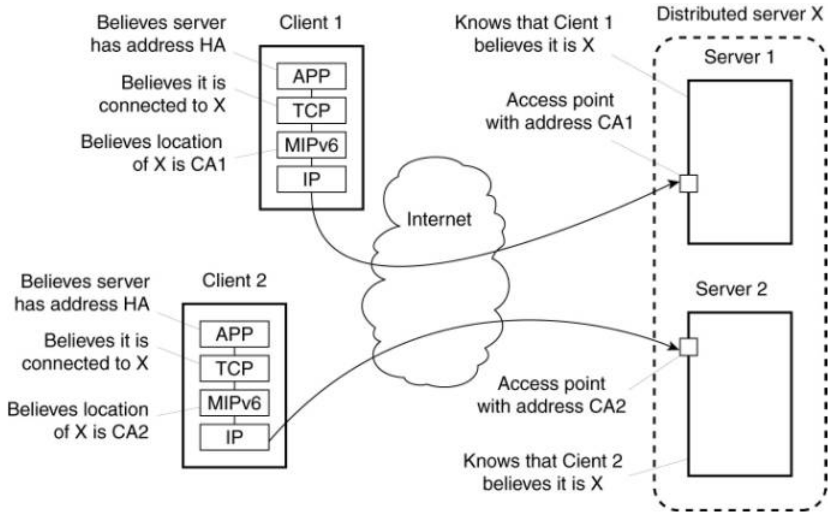
- Make use of available networking services - mobility support for IP version 6 (MIPv6)
    - MIPv6 -a mobile node is assumed to have a home network where it normally resides and for which it has an associated stable address, known as its home address (HoA).
    - Home network has a special router attached, known as the home agent, which will take care of traffic to the mobile node when it is away.
        - When a mobile node attaches to a foreign network, it will receive a temporary care-of address (CoA) where it can be reached.
        - This care-of address is reported to the node's home agent who will then see to it that all traffic is forwarded to the mobile node.
        - Note that applications communicating with the mobile node will only see the address associated with the node's home network.

- Use MIPv6 to offer a stable address of a distributed server.
    - A single unique contact address is initially assigned to the server cluster.
    - The contact address will be the server's life-time address to be used in all communication with the outside world.
    - At any time, one node in the distributed server will operate as an access point using that contact address, but this role can be taken over by another node.
    - The access point records its own address as the care-of address at the home agent associated with the distributed server.
    - At that point, all traffic will be directed to the access point, who will then take care in distributing requests among the currently participating nodes.
    - If the access point fails, a simple fail-over mechanism comes into place by which another access point reports a new care-of address.

- Simple configuration could create a bottleneck as all traffic would flow through these two machines.
    - Use MIPv6 feature known as route optimization.
        - Whenever a mobile node with home address HA reports its current care-of address, say CA, the home agent can forward CA to a client.
        - The client will then locally store the pair (HA, CA).
        - From that moment on, communication will be directly forwarded to CA.
        - Although the application at the client side can still use the home address, the underlying support software for MIPv6 will translate that address to CA and use that instead.

- Route optimization can be used to make different clients believe they are communicating with a single server, where, in fact, each client is communicating with a different member node of the distributed server (Figure).
    - When an access point of a distributed server forwards a request from client C1 to, say node S1 (with address CA1), it passes enough information to S1 to let it initiate the route optimization procedure by which eventually the client is made to believe that the care-of address is CA1. This will allow C1 to store the pair (HA, CA1).
    - During this procedure, the access point (as well as the home agent) tunnel most of the traffic between C1 and S1.
    - This will prevent the home agent from believing that the care-of address has changed, so that it will continue to communicate with the access point.

Route optimization in a distributed server.


Believes server has address HA
Client 1
Knows that Cient 1 believes it is X
Distributed server X
Server 1
APP
Believes it is connected to X
TCP
Access point with address CA1
Believes location of X is CA1
MIPv6
IP
Internet
Believes server has address HA
Client 2
Server 2
APP
Believes it is connected to X
TCP
MIPv6
Access point with address CA2
Believes location of X is CA2
IP
Knows that Cient 2 believes it is X

Managing Server Clusters

Common Approaches

1. Extend the traditional managing functions of a single computer to that of a cluster (most common approach).
   - an administrator can log into a node from a remote client and execute local managing commands to monitor, install, and change components.

2. Provide an interface at an administration machine that allows collection of information from one or more servers, upgrade components, add and remove nodes, etc.
   - This type of managing server clusters is widely applied in practice (e.g. Cluster Systems Management from IBM (Hochstetler and Beringer, 2004).

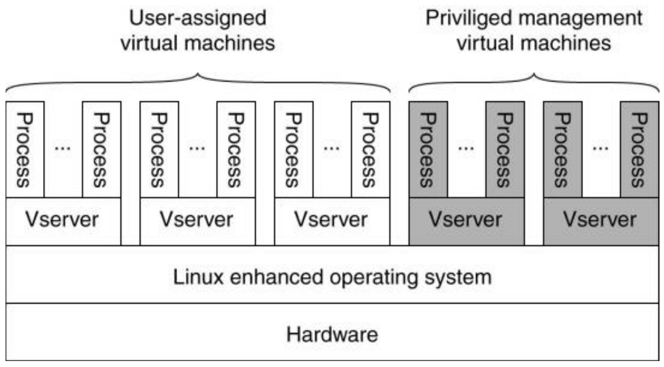
3. Ad hoc.
   - Managing thousands of servers, organized into many clusters but all operating collaboratively can not be done with centralized administration.
   - Very large clusters need continuous repair management (including upgrades).
     - If p is the probability that a server is currently faulty, and we assume that faults are independent, then for a cluster of N servers to operate without a single server being faulty is (1-p)N. With p=0.001 and N=1000, there is only a 36 percent chance that all the servers are correctly functioning.
   - Rules of thumb that should be considered (Brewer, 2001), but there is no systematic approach to dealing with massive systems management.
   - Cluster management is still in its infancy

Example: PlanetLab (Peterson et al., 2005).
- PlanetLab is a collaborative distributed system in which different organizations each donate one or more computers, adding up to a total of hundreds of nodes.
- These computers form a 1-tier server cluster, where access, processing, and storage can all take place on each node individually.
- Management of PlanetLab is almost entirely distributed.

In PlanetLab:
- An organization donates one or more nodes, where each node is easiest thought of as just a single computer, although it could also be itself a cluster of machines.
- Each node is organized as shown in figure.
- There are two important components (Bavier et al., 2004).
  1. The virtual machine monitor (VMM), which is an enhanced Linux operating system.
  2. vservers.
     - A (Linux) vserver is a separate environment in which a group of processes run.
       - Processes from different vservers are completely independent.
       - A vserver provides an environment consisting of its own collection of software packages, programs, and networking facilities.

- The Linux VMM ensures that vservers are separated: processes in different vservers are executed concurrently and independently, each making use only of the software packages and programs available in their own environment.
- PlanetLab introduces the notion of a slice - a set of vservers, each vserver running on a different node.
  - A slice is a virtual server cluster, implemented by means of a collection of virtual machines.

The basic organization of a PlanetLab node.


User-assigned virtual machines
Priviliged management virtual machines
Process
...
Process
Vserver
Linux enhanced operating system
Hardware

Issues managing PlanetLab:

1. Nodes belong to different organizations. Each organization should be allowed to specify who is allowed to run applications on their nodes, and restrict resource usage appropriately.

2. There are various monitoring tools available, but they all assume a very specific combination of hardware and software. Moreover, they are all tailored to be used within a single organization.

3. Programs from different slices but running on the same node should not interfere with each other. This problem is similar to process independence in operating systems.

Managing PlanetLab is done through intermediaries.

- One important class of such intermediaries is formed by slice authorities.
    - slice authorities have obtained credentials at nodes to create slices.
    - Obtaining these credentials has been achieved out-of-band, essentially by contacting system administrators at various sites.

- There are also management authorities.
    - A management authority is responsible for keeping an eye on nodes.
    - It ensures that the nodes under its regime run the basic PlanetLab software and abide to the rules set out by PlanetLab.
    - Service providers trust that a management authority provides nodes that will behave properly

Organization leads to the management structure shown below, described in terms of trust relationships

- The relations are as follows:

1. A node owner puts its node under the regime of a management authority, possibly restricting usage where appropriate.

2. A management authority provides the necessary software to add a node to PlanetLab.

3. A service provider registers itself with a management authority, trusting it to provide well-behaving nodes.

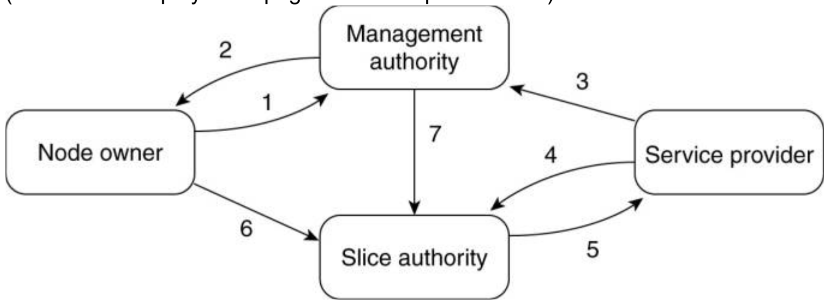
4. A service provider contacts a slice authority to create a slice on a collection of nodes.

5. The slice authority needs to authenticate the service provider.

6. A node owner provides a slice creation service for a slice authority to create slices. It essentially delegates resource management to the slice authority.

7. A management authority delegates the creation of slices to a slice authority.

The management relationships between various PlanetLab entities.
(This item is displayed on page 101 in the print version)



Issues: Monitoring.
Unified approach to allow users to see how well their programs are behaving within a specific slice.

- Every node is equipped with a collection of sensors, each sensor being capable of reporting information such as CPU usage, disk activity, and so on.
- Sensors can be arbitrarily complex, but the important issue is that they always report information on a per-node basis.
- This information is made available by means of a Web server: every sensor is accessible through simple HTTP requests).

Issues: protection of programs against each other
PlanetLab uses Linux virtual servers (called vservers) to isolate slices.

- A vserver is to run applications in there own environment, which includes all files that are normally shared across a single machine.
- Such a separation can be achieved relatively easy by means of the UNIX chroot command, which effectively changes the root of the file system from where applications will look for files.
- Only the superuser can execute chroot.

An overview of Linux virtual servers can be found in Potzl et al. (2005).

Code Migration

Approaches to Code Migration

Reasons for Migrating Code

Traditional method - process migration in which an entire process is moved from one machine to another (Milojicic et al., 2000).

1. Reason for doing so: overall system performance can be improved if processes are moved from heavily-loaded to lightly-loaded machines.
    - Load is expressed in terms of the CPU queue length or CPU utilization.

   BUT … Moving a running process to a different machine is a costly and intricate task.
    - Many modern distributed systems - optimizing computing capacity is less an issue than minimizing communication.
    - Platform and network heterogeneity make decisions for performance improvement through code migration based on qualitative reasoning instead of mathematical models.

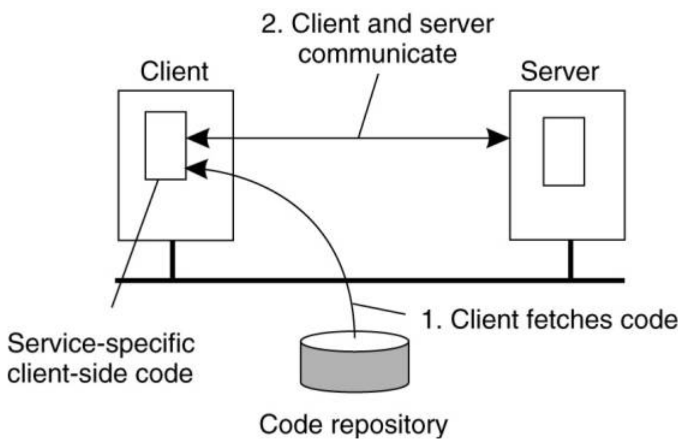
   Examples:
   Client-server system where server manages a huge database.
   - If a client application needs to perform many database operations involving large quantities of data, it may be better to ship part of the client application to the server and send only the results across the network.
   - Otherwise, the network may be swamped with the transfer of data from the server to the client. In this case, code migration is based on the assumption that it generally makes sense to process data close to where those data reside.

   Migrating parts of the server to the client.
    - Many interactive database applications clients need to fill in forms that are subsequently translated into a series of database operations.
    - Processing the form at the client side, and sending only the completed form to the server, can avoid that a relatively large number of small messages need to cross the network.
    - Result - the client perceives better performance, while at the same time the server spends less time on form processing and communication.

2. Reason for doing so: flexibility - It is possible to dynamically configure distributed systems.
   Example - Client / Server application
   - Traditional Implementation - server implements a standardized interface to a file system.
      i. Remote clients communicates with the server using a proprietary protocol.
      ii. The client-side implementation of the file system interface needs to be linked with the client application.
      iii. Approach requires the software be readily available to the client at the time the client application is being developed.

   - Alternative Implementation - server provides the client's implementation no sooner than is necessary - when the client binds to the server.
      i. The client dynamically downloads the implementation, goes through the necessary initialization steps, and invokes the server.
      ii. This model of dynamically moving code from a remote site does require that the protocol for downloading and initializing code is standardized.
      iii. The downloaded code must be executed on the client's machine.

Figure
The principle of dynamically configuring a client to communicate to a server.
The client first fetches the necessary software, and then invokes the server.


2. Client and server communicate
Client
Server
1. Client fetches code
Service-specific client-side code
Code repository

Advantages –

1. Clients need not have all the software preinstalled to talk to servers.
   The software can be moved as required, and discarded when no longer needed.
2. With standardized interfaces the client-server protocol and its implementation can be changed at will.
   Changes will not affect existing client applications that rely on the server.

Disadvantages -

Security

Blindly trusting that the downloaded code implements only the advertised interface while accessing an unprotected hard disk.

Models for Code Migration

Framework described in (Fuggetta et al. 1998)

A process consists of three segments:

1. Code segment - contains the set of instructions that make up the program that is being executed.
2. Resource segment - contains references to external resources needed by the process (e.g. files, printers, devices, other processes, etc.)
3. Execution segment - stores the current execution state of a process, consisting of private data, the stack, and the program counter.

Types of Process Mobility:

Weak mobility - Transfer only the code segment, along with some initialization data.

- Feature: a transferred program is always started from one of several predefined starting positions.
  - e.g., Java applets start execution from the beginning.
- Benefit: simplicity.
  - Weak mobility requires only that the target machine can execute that code

Strong mobility - execution segment is transferred as well.

- Feature: a running process can be stopped, moved to another machine, and resume execution where it left off.
  - More general than weak mobility, but more difficult to implement.
- Can also be supported by remote cloning
  - Cloning yields an exact copy of the original process, but running on a different machine.
  - The cloned process is executed in parallel to the original process.
  - In UNIX systems, remote cloning takes place by forking off a child process and letting that child continue on a remote machine.
  - The benefit of cloning is that the model closely resembles the one that is already used in many applications.
  - Migration by cloning is a simple way to improve distribution transparency.

Migration Initiation:

Sender-initiated - migration is initiated at the machine where the code currently resides or is being executed.

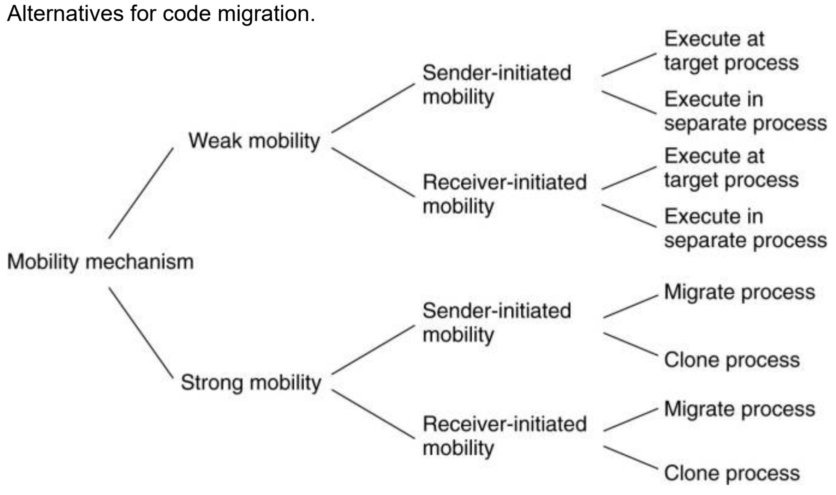
- Done when uploading programs to a compute server.
- Securely uploading code to a server often requires that the client has previously been registered and authenticated at that server.

Receiver-initiated - Initiative for code migration is taken by the target machine.

- Java applets are an example.
- Receiver-initiated migration is simpler than sender-initiated migration.
  - Code migration usually occurs between a client and a server, where the client takes the initiative for migration.
  - Downloading code can often be done anonymously.
  - The server is not interested in the client's resources.

- Code migration to the client is done only for improving clientside performance.
- Only a limited number of resources need to be protected, such as memory and network connections.

Alternatives for code migration.


Mobility mechanism
Weak mobility
Strong mobility
Sender-initiated mobility
Receiver-initiated mobility
Sender-initiated mobility
Receiver-initiated mobility
Execute at target process
Execute in separate process
Execute at target process
Execute in separate process
Migrate process
Clone process
Migrate process
Clone process

Migration and Local Resources

Issue: Migrating Resource Segment
Resource segment cannot always be transferred along with the other segments without being changed

- Example: a process holding a reference to a specific TCP port through which it was communicating with other (remote) processes.
  - This reference is held in its resource segment.
  - When the process moves to another location, it must release the port and request a new one at the destination.

Types of process-to-resource bindings: (Fuggetta et al. 1998)

1. Binding by identifier - strongest binding - when a process refers to a resource by its identifier.
   - The process requires precisely the referenced resource, and nothing else.
   - Example : a process uses a URL to refer to a specific Web site or when it refers to an FTP server by means of that server's Internet address.

2. Binding by value - weaker binding - when only the value of a resource is needed.
   - Process execution is not affected if another resource provided that same value.
   - Example: a program relies on standard libraries, such as those for programming in C or Java.
     - Such libraries should always be locally available, but their exact location in the local file system may differ between sites.
     - Not the specific files, but their content is important for the proper execution of the process.

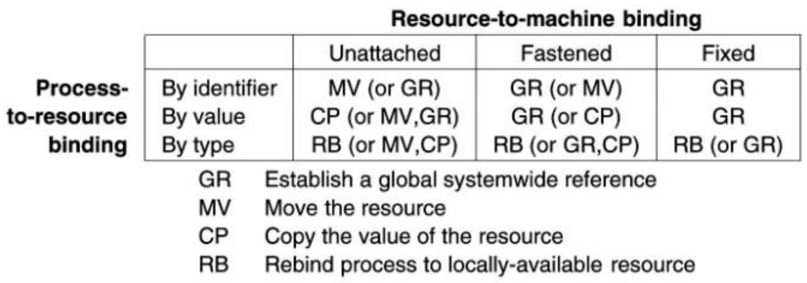
3. Binding by type – weakest binding - when a process indicates it needs only a resource of a specific type.
   - Exemplified by references to local devices, such as monitors, printers, etc.

Issue: Code migration requires consideration of the resource-to-machine bindings
Cases:

1. Unattached resources can be easily moved between different machines
   i. Typically (data) files associated only with the program that is to be migrated.
2. Moving or copying a fastened resource may be possible, but only at relatively high costs.
   i. Typical examples of fastened resources are local databases and complete Web sites.
   - Although such resources are, in theory, not dependent on their current machine, it is often infeasible to move them to another environment.
3. Fixed resources are intimately bound to a specific machine or environment and cannot be moved.
   i. Fixed resources are often local devices or a local communication end point.

Combining three types of process-to-resource bindings, and three types of resource-to-machine bindings, leads to nine combinations that we need to consider when migrating code.

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



		Resource-to-machine binding		
		Unattached	Fastened	Fixed
Process-to-resource binding	By identifier	MV (or GR)	GR (or MV)	GR
	By value	CP (or MV,GR)	GR (or CP)	GR
	By type	RB (or MV,CP)	RB (or GR,CP)	RB (or GR)

GR	Establish a global systemwide reference
MV	Move the resource
CP	Copy the value of the resource
RB	Rebind process to locally-available resource

Examples:
A process is bound to a resource by identifier.

i. When the resource is unattached, it is best to move it along with the migrating code.
ii. When the resource is shared by other processes establish a global reference
    - a reference that can cross machine boundaries.
        - e.g. a URL.
iii. When the resource is fastened or fixed, the best solution is to create a global reference.

Problems with global reference
Reference may be expensive to construct.
e.g. a program that generates high-quality images for a dedicated multimedia workstation.

- Fabricating high-quality images in real time is a compute-intensive task, for which reason the program may be moved to a high-performance compute server.
- Establishing a global reference to the multimedia workstation means setting up a communication path between the compute server and the workstation.
- Significant processing involved at both the server and the workstation to meet the bandwidth requirements of transferring the images.
- Nt result - moving the program to the compute server is not such a good idea, only because the cost of the global reference is too high.

Migrating a process that makes use of a local communication end point.

- Here is a fixed resource to which the process is bound by the identifier.
- Two solutions:
    1. Let the process set up a connection to the source machine after it has migrated and install a separate process at the source machine that forwards all incoming messages.
        - Drawback - whenever the source machine malfunctions, communication with the migrated process may fail.
    2. Have all processes that communicated with the migrating process, change their global reference, and send messages to the new communication end point at the target machine.

e.g. bindings by value.

- A fixed resource.
    - The combination of a fixed resource and binding by value occurs when a process assumes that memory can be shared between processes.
    - Establishing a global reference would require a distributed form of shared memory.
    - In many cases, this is not really a viable or efficient solution.

- Fastened resources.
    - Fastened resources that are referred to by their value, are typically runtime libraries.
    - Normally, copies of such resources are readily available on the target machine, or should otherwise be copied before code migration takes place.
    - Establishing a global reference is a better alternative when huge amounts of data are to be copied, as may be the case with dictionaries and thesauruses in text processing systems.

- Unattached resources.
    - The best solution is to copy (or move) the resource to the new destination, unless it is shared by a number of processes.
    - In the latter case, establishing a global reference is the only option.

e.g. bindings by type.

- Irrespective of the resource-to-machine binding, the solution is to rebind the process to a locally available resource of the same type.
- If the resource is not available, must copy or move the original one to the new destination, or establish a global reference.

Migration in Heterogeneous Systems

- Distributed systems are constructed on a heterogeneous collection of platforms, each having their own operating system and machine architecture.
- Migration requires that each platform be supported - the code segment can be executed on each platform.
- Must ensure that the execution segment can be properly represented at each platform.

Problems of heterogeneity same as those of portability.

- 1970's - solution to alleviate problems of porting Pascal to different machines was to generate machine-independent intermediate code for an abstract virtual machine that would be implemented on many platforms, but it would then allow Pascal programs to be run anywhere.
- 25 Years Later
    - Code migration in heterogeneous systems attacked by scripting languages and highly portable languages such as Java.
        - These solutions adopt the same approach as was done for porting Pascal.
        - All rely on a (process) virtual machine that either directly interprets source code (as in the case of scripting languages), or otherwise interprets intermediate code generated by a compiler (as in Java).

Migrate entire computing environments

- Compartmentalize the overall environment to provide processes in the same part their own view on their computing environment.
- Can decouple a part from the underlying system and migrate it to another machine.
- Migration would provide a form of strong mobility for processes, as they can then be moved at any point during their execution, and continue where they left off when migration completes.
- Solves bindings to local resources problem - local resources become part of the environment that is being migrated.

Why migrate entire environments?

It allows continuation of operation while a machine needs to be shutdown.

- e.g. in a server cluster, the systems administrator may decide to shut down or replace a machine
    - can temporarily freeze an environment, move it to another machine (where it sits next to other, existing environments), and unfreeze it.

Example: Real-time migration of a virtualized operating system (Clark et al. 2005).

- Useful in a cluster of servers where a tight coupling is achieved through a single, shared local-area network.
- Migration involves two major problems:
    - real-time migration of a virtualized operating system
    - migrating bindings to local resources.

Real-time migration of a virtualized operating system:

Three ways to handle migration (which can be combined):

1. Pushing memory pages to the new machine and resending the ones that are later modified during the migration process.
2. Stopping the current virtual machine; migrate memory, and start the new virtual machine.
3. Letting the new virtual machine pull in new pages as needed, that is, let processes start on the new virtual machine immediately and copy memory pages on demand.

- Option #2 may lead to unacceptable downtime if the migrating virtual machine is running a live service.
- Option #3 extensively prolong the migration period lead to poor performance because it takes a long time before the working set of the migrated processes has been moved to the new machine.

- Clark et al. propose to use a pre-copy approach which combines the Option #1, along with a brief stop-and-copy phase as represented by the Option #2.
    - This combination can lead to service downtimes of 200 ms or less.