Processes
Overview

▶ Threads vs Processes
▶ Clients
▶ Servers
▶ Virtualization
▶ Code Migration
Threads versus Processes

*Threads*:  
- Improve application responsiveness by allowing them to not block  
- Allows for parallel computation resulting in higher speed on many-core systems  
- Easier to structure many applications as a collection of cooperating threads  
- Higher performance compared to multiple processes since switching between threads takes less time

*Processes*:  
- Simpler to synchronize as processes have separate memory space  
- Allows for parallel computation resulting in higher speed on many-core systems
Thread Implementations

- **User-space threads:**
  - Creating/destroying threads is inexpensive
  - Switching context is very fast
  - But invocation of a blocking system call blocks all threads...
  - Cannot make use of multiple cores

- **Kernel threads:**
  - Overcomes the last two issues with user-space threads but loses performance

- **Light-Weight Processes (LWP):** Hybrid model. Multiple LWP/threads run inside a single (heavy-weight) process. In addition, the system offers a user-level threads package.
A server is a process implementing a specific service on behalf of a collection of clients.

A server can be iterative or concurrent. Concurrent servers can be multi-threaded or multi-process.

Characteristics of server implementations:

<table>
<thead>
<tr>
<th>Model</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-threaded</td>
<td>No parallelism, blocking system calls</td>
</tr>
<tr>
<td>Multi-threaded</td>
<td>Parallelism, blocking system calls</td>
</tr>
<tr>
<td>Finite state machine</td>
<td>Parallelism, nonblocking system calls</td>
</tr>
</tbody>
</table>

See example LargerHttpd.java in folder sockets in package tcp.largerhttpd.
How does a client find a server? Need to know the end-point or port and the host address.

- Statically assigned like well known servers like HTTP on port 80.
- Look-up service provided by a special directory/registry server.
- Using a superserver that selects on multiple ports and forks off the appropriate server when a request comes in.
**SuperServer Setup**

**Diagram:**
- **Client machine:**
  - Client
  - 1. Request service
- **Server machine:**
  - Actual server
  - Super-server
  - 2. Continue service
- **Flow:**
  - Client requests service
  - Super-server handles request
  - Actual server provides service
  - Client continues service
- **Server function:**
  - Create server for requested service

---

**Text:**

`xinetc`d is an example of a superserver.
Design Issues for Servers (3)

- **Stateless** server: A stateless server does not remember anything from one request to another. For example, a HTTP server is stateless. **Cookies** can be used to transmit information specific to a client with a stateless server. Easy to recover from a crash.

- **Stateful** server: Maintains information about its clients. Performance improvement over stateless servers is often the reason for stateful servers. Needs to recover its entire state as it was just before crash. Can be quite complex for distributed servers.

- **Soft State** servers: The server promises to maintain state on behalf of the client, but only for a limited time.
How to handle communication interrupts? Use out-of-band data. Example: to cancel the upload of a huge file.

- Server listens to separate endpoint, which has higher priority, while also listening to the normal endpoint (with lower priority).
- Send urgent data on the same connection. Can be done with TCP, where the server gets a signal (SIGURG) on receiving urgent data.
In-class Exercises

▶ Sketch the design of a multithreaded server that supports multiple protocols using sockets as its transport-level interface to the underlying operating system.

▶ Is a server that maintains a TCP/IP connection to a client stateful or stateless?

▶ Imagine a web server that maintains a table in which client IP addresses are mapped to the most recently accessed web pages. When a client connects to the webserver, the server looks up the client in its table, and if found, returns the registered page. Is the server stateful or stateless?
Design of a Three-tiered Server Cluster
Server Clusters (2)

TCP Handoff (uses IP forwarding and IP spoofing)
Server Clusters (3)

- Switch can hand off connections in round-robin and thus be oblivious of the service being provided.
- Switch can handoff request based on type of service requested to the appropriate server.
- Switch can handoff request based on server loads.
- Switch can handoff request by being aware of the content.
- Single point of access can be made better using DNS to map one hostname to several servers. But the client still has to try multiple servers in case some are down.
Distributed Servers

- A **distributed server** is a possibly dynamically changing set of machines, with also possibly varying access points, but which nevertheless appears to the outside world as a single, powerful machine.

- A stable access point across a distributed server can be implemented using mobility support for IPv6 (MIPv6).
  
  - **home network**: where a mobile node normally resides
  - **home address (HoA)**: stable address for a node in its home network
  - **care of address (CoA)**: when a mobile node attaches to a foreign network, it sets up a forwarding from its HoA to the CoA
  - **route optimization** from MIPv6 is used to make different clients believe they are communicating with a single server where, in fact, each is communicating with a different member node of the distributed server
Virtualization (1)

- Virtualization is the creation of a virtual version of something, such as a hardware platform, operating system, a storage device or network resources.
Virtualization (2)

Benefits of virtualization:

▶ Allows software at higher level (like middleware and applications) to be supported on changing hardware and lower level systems software.
▶ Eases administration of large number of servers (or resources).
▶ Helps with scalability and better utilization of hardware resources.
▶ The main driver behind the growth in cloud computing and utility computing.
An interface between the hardware and software consisting of machine instructions that can be invoked by any program.

An interface between the hardware and software, consisting of machine instructions that can be invoked only by privileged programs, such as an operating system.

An interface consisting of system calls as offered by an operating system.

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.
Architectures for Virtual Machines (2)
Process Virtual Machine: An abstract instruction set that is to be used for executing applications. For example: Java Virtual Machine.
Virtual Machine Monitor: A layer completely shielding the original hardware but offering the complete instruction set of that same (or other hardware) as an interface.

Makes it possible to have multiple instances of different operating systems run simultaneously on the same platform.

Monolithic Examples: VMWare Workstation, VMWare Fusion, Virtual Box, Parallels, Windows Virtual PC, etc

Microkernel Examples: Hyper-V, VMWare ESX/ESXi, Xen, z/VM.
Architectures for Virtual Machines (5)

- **Sandbox** (Application-level). Examples: Citrix XenApp, ZeroVM.

- **Containers** (Environment-level). Examples: *cgroups*-based Docker and LXC (Linux Containers). Lighter weight compared to Virtual Machine Monitors.
Cloud Computing Layers

**IaaS**: Amazon Web Services, Microsoft Azure

**PaaS**: Google App Engine, Microsoft Azure

**SaaS**: CRM, Email, virtual desktop, communication, games, ...

**PaaS**: Execution runtime, database, web server, development tools, ...

**IaaS**: Virtual machines, servers, storage, load balancers, network, ...
Code Migration

- Improve computing performance by moving processes from heavily-loaded machines to lightly loaded machines.
- Improve communication times by shipping code to systems where large data sets reside. E.g. a client ships code to a database server or vice versa.
- **Mobile Agents**: small piece of code that moves from site to site for a web search. Several copies can be made to improve performance.
- **Flexibility to dynamically configure distributed systems**. E.g. a server can provide interface code to a client dynamically. This does require that the protocol for downloading and initializing the code is standardized. Allows the interface to be changed as often as desired without having to rebuild applications or servers.
Recall that a process consists of three segments: \textit{code segment}, \textit{resource segment}, \textit{execution segment}.

- **Weak mobility**: Only the code segment (and some initialization data) can be transferred. Transferred program always starts at one of the predefined starting positions.

- **Strong mobility**: Code and execution segments can both be transferred.

- **Sender-initiated versus receiver-initiated**: Uploading code to a server versus downloading code from a server by a client.
Code Migration Models (2)

Mobility mechanism

Weak mobility
- Sender-initiated mobility
  - Execute at target process
  - Execute in separate process
- Receiver-initiated mobility
  - Execute at target process
  - Execute in separate process

Strong mobility
- Sender-initiated mobility
  - Migrate process
  - Clone process
- Receiver-initiated mobility
  - Migrate process
  - Clone process
Three ways to handle migration (which can be combined)

- **Binding by identifier**: A process requires precisely the referenced resource. For example using a URL for a web site or a FTP server.

- **Binding by value**: The process can continue to execute if it gets the same value from another resource. E.g. A standard library.

- **Binding by type**: The process only needs a resource of specific type. E.g. a printer.
Three ways to handle migration (which can be combined)

- **Unattached resources**: can be easily moved between machines. E.g. Data files.
- **Fastened resource**: Can only be moved at a high cost. E.g. a local database, a web site.
- **Fixed resource**: Intimately bound to a specific machine and environment and cannot be moved. E.g. Local devices, local communication end point (a socket bound to an address and port).
## Migration and Local Resources

<table>
<thead>
<tr>
<th>Process-to-resource binding</th>
<th>Unattached</th>
<th>Fastened</th>
<th>Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>By identifier</td>
<td>MV (or GR)</td>
<td>GR (or MV)</td>
<td>GR</td>
</tr>
<tr>
<td>By value</td>
<td>CP (or MV,GR)</td>
<td>GR (or CP)</td>
<td>GR</td>
</tr>
<tr>
<td>By type</td>
<td>RB (or MV,CP)</td>
<td>RB (or GR,CP)</td>
<td>RB (or GR)</td>
</tr>
</tbody>
</table>

- GR: Establish a global systemwide reference
- MV: Move the resource
- CP: Copy the value of the resource
- RB: Rebind process to locally-available resource
Consider a process $P$ that requires access to file $F$, which is locally available on the machine that $P$ is currently available. When $P$ moves to another machine, it still requires access to $F$. If the file-to-machine binding is fixed, how could the systemwide reference to $F$ be implemented.

Is your chat server implementation *stateless* or *stateful*? Discuss.

Is your filesync server implementation *stateless* or *stateful*? Discuss.

Read the Wikipedia article on *cloud computing*.

Read the Wikipedia article on *proxy server*. 