Outline

- Processes
  - Multithreading
  - Virtualization
  - Clients
  - Servers

- Q&A
Thread Usage in Nondistributed Systems

Context switching as the result of IPC

S1: Switch from user space to kernel space

S2: Switch context from process A to process B

S3: Switch from kernel space to user space

Context switch requires changing the memory map in the MMU (Memory Management Unit) and flushing the TLB (Translation Lookaside Buffer)
Thread Implementation

- **User-Level Threads**
  - Being Executed Entirely in User Space
    - It is cheap to create and destroy threads
    - Switching can often be done efficiently
    - Invocation of a blocking system call blocks all
  - Having the Kernel Aware of Threads and Schedule Them
    - Switching may become expensive

- **Lightweight Process (LWP)**
  - Hybrid Form of User and Kernel-Level Threads
    - Each LWP can run its own (user-level) thread
    - Synchronization does not require kernel support
Thread Implementation (Cont’d)

Combining kernel-level lightweight processes and user-level threads

Tread doing a blocking system call still continues in the context of the current LWP. In switching, the selected LWP will simply continue where it had previously left off.
Multithreaded Servers

A multithreaded server organized in a dispatcher/worker model
### Multithreaded Servers (Cont’d)

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

The process is being operated as a finite state machine that gets an event and then react to it, depending on what is in it; this approach performs well, but is hard to program.

**Three ways to construct a server**
The Role of Virtualization in Distributed Systems

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

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved
Architectures of Virtual Machines

- Interfaces at Different Levels
  - 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
Architectures of Virtual Machines (Cont’d))

- Interfaces at Different Levels (Cont’d)
  - 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 of Virtual Machines (Cont’d)

Various interfaces offered by computer systems
Architectures of Virtual Machines (Cont’d)

(a) A process virtual machine, with multiple instances of (application, runtime) combinations

Instructions can be interpreted (as is the case for the Java executing applications), but could also be emulated mimicking the behavior of system calls.
Architectures of Virtual Machines (Cont’d)

(b) A virtual machine monitor, with multiple instances of (applications, operating system) combinations
Networked User Interfaces

(a) A networked application with its own protocol
Networked User Interfaces (Cont’d)

(b) A general solution to allow access to remote applications

(b) Thin-Client Approach

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved
Example: The X Window System

The basic organization of the X Window System
Client-Side Software for Distribution Transparency

Client machine

Server 1

Server 2

Server 3

Client side handles request replication

Replicated request

Transparent replication of a server using a client-side solution

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved
General Design Issues

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

The general organization of a three-tiered server cluster

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved
Server Clusters (Cont’d)

Logically a single TCP connection

Request

Response

Request (handed off)

The principle of TCP handoff

Server

Server
Route optimization in a distributed server