Chapter 3
Processes
Thread Usage in Nondistributed Systems

Figure 3-1. Context switching as the result of IPC.
Thread Implementation

Figure 3-2. Combining kernel-level lightweight processes and user-level threads.
Multithreaded Servers (1)

Figure 3-3. A multithreaded server organized in a dispatcher/worker model.
Multithreaded Servers (2)

<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>

Figure 3-4. Three ways to construct a server.
The Role of Virtualization in Distributed Systems

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

Figure 3-5.
Architectures of Virtual Machines (1)

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 (2)

Interfaces at different levels

- 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 (3)

Figure 3-6. Various interfaces offered by computer systems.
Architectures of Virtual Machines (4)

Figure 3-7. (a) A process virtual machine, with multiple instances of (application, runtime) combinations.
Architectures of Virtual Machines (5)

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

Figure 3-7.
Networked User Interfaces (1)

(a)

Figure 3-8. (a) A networked application with its own protocol.
Figure 3-8. (b) A general solution to allow access to remote applications.
Example: The XWindow System

Figure 3-9. The basic organization of the XWindow System.
Thin Clients

Solutions to X11 Performance Problems

• Re-engineer the implementation
  • Compression
  • Caching

• Simplify the display server
  • Entire display is controlled on the application side
  • Requires sophisticated compression

• Simplify the display protocol
Client-Side Software for Distribution Transparency

![Diagram showing client-side software for distribution transparency]

Figure 3-10. Transparent replication of a server using a client-side solution.
General Design Issues (1)

Client machine

Client

2. Request service

Server machine

Server

1. Ask for end point

Daemon

Register end point

End-point table

(a)

Figure 3-11. (a) Client-to-server binding using a daemon.
General Design Issues (2)

Figure 3-11. (b) Client-to-server binding using a superserver.
Server Clusters (1)

Figure 3-12. The general organization of a three-tiered server cluster.

Tannenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, © 2007 Prentice-Hall, Inc. All rights reserved.
Server Clusters (2)

Logically a single TCP connection

Request

Response

Request (handed off)

Server

Figure 3-13. The principle of TCP handoff.
Figure 3-14. Route optimization in a distributed server.
Managing Server Clusters

Example: PlanetLab

User-assigned virtual machines

Priviliged management virtual machines

Process ... Process ... Process ... Process ... Process ... Process ... Process ... Process ... Process

Vserver Vserver Vserver Vserver Vserver Vserver Vserver Vserver

Linux enhanced operating system

Hardware

Figure 3-15. The basic organization of a PlanetLab node.
PlanetLab (1)

PlanetLab management issues:

- 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.
- Monitoring tools available assume a very specific combination of hardware and software.
  - All tailored to be used within a single organization.
- Programs from different slices but running on the same node should not interfere with each other.
PlanetLab (2)

Figure 3-16. The management relationships between various PlanetLab entities.
PlanetLab (3)

Relationships between PlanetLab entities:

• A node owner puts its node under the regime of a management authority, possibly restricting usage where appropriate.
• A management authority provides the necessary software to add a node to PlanetLab.
• A service provider registers itself with a management authority, trusting it to provide well-behaving nodes.
PlanetLab (4)

Relationships between PlanetLab entities:

- A service provider contacts a slice authority to create a slice on a collection of nodes.
- The slice authority needs to authenticate the service provider.
- A node owner provides a slice creation service for a slice authority to create slices. It essentially delegates resource management to the slice authority.
- A management authority delegates the creation of slices to a slice authority.
Reasons for Migrating Code

1. Client fetches code
2. Client and server communicate

Service-specific client-side code
Code repository

Figure 3-17. The principle of dynamically configuring a client to communicate to a server. The client first fetches the necessary software, and then invokes the server.
Models for Code Migration

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

Figure 3-18. Alternatives for code migration.
## Migration and Local Resources

### Resource-to-machine binding

<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

**Figure 3-19.** Actions to be taken with respect to the references to local resources when migrating code to another machine.
Migration in Heterogeneous Systems

Three ways to handle migration (which can be combined)

- Pushing memory pages to the new machine and resending the ones that are later modified during the migration process.
- Stopping the current virtual machine; migrate memory, and start the new virtual machine.
- 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.