OS Mechanisms
Posix 1003.1c

Synchronization

- Mutex
  - Generally a separate mutex variable is used for each variable to be protected.
  - pthread_mutex_init
  - pthread_mutex_lock
  - pthread_mutex_unlock

- Semaphores
  - Generally one semaphore for each resource with count N.
  - sem_init
  - sem_wait
  - sem_post

- Condition Variables
  - A tool for applications in need of busy wait.

Concurrent Execution

- All threads appear to execute at the same time.

- Each thread has its own stack.

- Shared Globals
  - All threads within the same process share the same global variables.

- Shared File Descriptions
  - All threads within the same process share the file descriptors.

Issues in Client Design-1

- Choosing A Local Port Number
  - A client can choose any port number.
  - No conflict with other port numbers in use.
  - Not a well known server port number.
  - If a process does not explicitly call bind, connect or listen, which ever is called kernel automatically picks a valid ephemeral port and source IP.

- Choosing A Local IP address
  - In a multi-homed computer applications may not know which IP adapter is in use.
  - While binding, setting IP address to 0(INADDR_ANY) will allow the kernel to pick the correct IP address.

Issues in Client Design-2

- Reading from a Stream
  - Only one write operation may require multiple reads at the other end.
  - The flag can be 0 or AND of few values such as:
    - MSG_PEEK: look at the data that is available to read, but without having system discard it from system buffer.
    - MSG_WAITALL: do not return until the specified amount of data is received.

```c
n = recv(socket, buf, sizeof(buf), 0);
while (n > 0) {
    write(1, buf, n);
    n = recv(socket, buf, sizeof(buf), 0);
}
```
**Issues in Server Design-1**

- Binding to a Well Known Port
  - If the service has to be available to a wide audience, the server must bind to a well-known port.
- Setting up the IP address
  - Like clients, a server may leave it up to the kernel to set the IP address to avoid confusion in multi-homed hosts.

**Concurrent Server**

- Fork
- Listen
- Connect Request
- TCP

Now, sd3 is free to receive requests from other clients.

TCP Port Numbers and Concurrent Servers

<table>
<thead>
<tr>
<th>Client1</th>
<th>Server</th>
<th>Client2</th>
</tr>
</thead>
<tbody>
<tr>
<td>198.69.10.2.1</td>
<td>198.69.10.2.10</td>
<td>198.69.10.2.21</td>
</tr>
<tr>
<td>(206.62.226.35, 206.62.226.35.21)</td>
<td>(206.62.226.35.21, 206.69.10.2.1901)</td>
<td>(206.62.226.35.21, 206.62.226.35.21)</td>
</tr>
</tbody>
</table>

TCP identifies a connection with all the 4 values and delivers the data accordingly.

**Connection Oriented & Connectionless Communication**

- UDP Client
- UDP Server

**Issues in Server Design-2**

- Concurrent vs. Iterative server
  - Simplicity, response time
  - Real vs. apparent concurrency
- Connection-Oriented vs. Connectionless Access
  - Reliability, persistence, data volume, flow-control

<table>
<thead>
<tr>
<th>Iterative</th>
<th>Iterative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connectionless</td>
<td>Connectionless</td>
</tr>
<tr>
<td>Concurrent</td>
<td>Concurrent</td>
</tr>
<tr>
<td>connectionless</td>
<td>connectionless</td>
</tr>
</tbody>
</table>
Concurrency Management

Perspectives:
- User perspective: response time
- System perspective: impact on resources

Issues:
- How can programmer know whether concurrency is warranted?
- How to determine which design is optimal?
- How can a programmer estimate demand or service time?

Concepts:
- Level of Concurrency
- Demand-Driven Concurrency

Cost of Concurrency

+ Overhead and Delay

Iterative

\[
\begin{array}{c|cc}
\text{Process req.1} & \text{Process req.2} \\
0 & p & 2p \\
\end{array}
\]

Concurrent

\[
\begin{array}{c|cc}
\text{Create slave 1} & \text{Create slave 2} \\
0 & c & 2c & 2c+p \\
\end{array}
\]

Quiz: if the rate at which requests arrives exceeds 1/c but is less than 1/p which implementation can handle the load?

Process Preallocation

- Demand-driven concurrency can be avoided by limiting the maximum level of concurrency.
- To avoid the run time process creation delay, preallocate service processes.
- Design:
  - Master server creates N slaves at the beginning
  - Each slave waits/sleeps using OS support
  - When service request arrives each slave by turn picks it.
  - When done slaves do not exit.
  - Preallocation allows the server process to switch and move to next process faster.

Delayed Process Allocation

- Standby slave approach solve runtime process creation
- But they too costs in terms of OS resource management.
- Iterative server can yield faster service and higher throughput if the service time is small.
- But, how can programmer know the service time?
- Design:
  - The server estimates the processing time dynamically by looking into service parameters (I/O size/bound).
  - Server starts processing a new request iteratively
  - And starts a timer
  - On time out, it invokes a concurrent slave.

Preallocation in a Connection Oriented Server

Preallocation in a Connectionless Server

Slaves use the same descriptor to listen for request and send data. Requests arrive in UDP. Descriptors are automatically freed after each communication.
Combined Technique

- Design:
  - server starts without any slave.
  - creates a slave only when timer expires.
  - but, once a slave has been created, it does not exit immediately.
  - Slaves can also spawn slaves, if needed.
- How to control concurrency?
  - master specifies limit of concurrency MAX to slaves.
  - Slave exits after a specified period of inactivity.

Multi Service Servers

- A typical host may run multiple services. One server can be designed to provide a set of services instead of one.
- Advantages:
  - easy management
  - communication code can be reused.
  - less impact on system resources.
- Design:
  - requires service descriptor code.
  - server use descriptor code based switch.
  - single process approach
  - concurrent approach
  - super server (UNIX INETD)

Advantages of Concurrency to Servers

- Improves observed response time.
- Thus it can increase client throughput.
- It can eliminate potential deadlocks and some denial of service attacks.
- It makes server design modular.
- Concurrent execution can easily be ported on multiprocessor hardware.

Concurrency in Client Design

- Concurrency makes client design simpler and modular too.
- Concurrent client can contact several servers without being held by one.
- Concurrency can allow user to change parameters, inquire about client parameters, or control client processing dynamically.

Multi Protocol Servers

- One server can talk via several underlying transport layer network protocols.
- Example: DAYTIME server maintained in UNIX systems can talk via TCP and UDP both
- Advantages:
  - easy software maintenance
  - easy debugging
  - service code can be reused.
  - less impact on system resources.
- Disadvantage:
  - lack of flexibility for network administrator
- Design:
  - multiple protocol specific listening ports
  - iterative polling
Design of Concurrent Client

- If the server sending data blocks, it can still keep on sending data to server in other direction.

Web Servers

- Client Request Handling
  - Read and parse the HTTP request message
  - Translate the URL to the file name
  - Determine if the request is authorized.
  - Generate and transmit the response.

Case Study: Concurrency in Apache

<table>
<thead>
<tr>
<th>Directive</th>
<th>Definition (default)</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartServers</td>
<td>Initial number of child processes (5)</td>
</tr>
<tr>
<td>MaxClients</td>
<td>Maximum number of child processes (250)</td>
</tr>
<tr>
<td>MaxClientsPerServer</td>
<td>Target maximum number of idle children (5)</td>
</tr>
<tr>
<td>MaxClientsPerClass</td>
<td>Target maximum number of idle children (10)</td>
</tr>
<tr>
<td>MaxRequestsPerChild</td>
<td>Maximum number of requests per child (50)</td>
</tr>
<tr>
<td>ListenBacklog</td>
<td>Maximum number of pending connections (511)</td>
</tr>
<tr>
<td>SendBufferSize</td>
<td>Size of the TCP sendBuffer (16 KB default)</td>
</tr>
<tr>
<td>KeepAliveRequests</td>
<td>Maximum number of requests per connection (10)</td>
</tr>
<tr>
<td>KeepAliveTimeout</td>
<td>Maximum idle time for connection (15 sec)</td>
</tr>
</tbody>
</table>

Web Server

- Client Request Handling
- Resource & Access Control
- Dynamical Response Handling
- Performance Enhancement

Client Request Handling

- Authentication
  - Identify the user
  - Must be performed for each request.
  - Browser typically helps user.
  - Users can be lumped in user groups.
- Authorization
  - Which user has access to which resource.
  - Policy is written in a “Access Control List”.
  - Can create severe server load, as “subdirectories” exist.

Access Control
Dynamic Response Handling

- Server Side Include
  - `<script type="text/javascript"> etc.`
  - Requires every HTML does to be parsed and scanned.
  - Use of filename extension `.js`.
  - Use of PHP (Personal Home Page) starts a PHP processor.
  - Use of Active Server Pages.

- Server Scripts
  - Not the HTML, but the URL points to a script, CGI etc.
  - Software module in same process.
  - Persistent process contacted by server.

Cookies

- In index to a particular user which can be used by server to identify an user across connection, or even user session.
- Reduces data volume as all user data can be stored in a server side database and cookies can be used as index.
- Also a threat to user privacy.
- First introduced in 1994 in Netscape 4.0; standardized in October 2000 in RFC 2965.
- Typical default: each site up to 4KB, maximum of 20 Cookies per server, and total of 500 per browser.

Performance Enhancement

- Store in memory the popular files and save disk reading time.
  - How to maintain consistency?
    - Write changes the morning must be invalidated.
    - Check last-update time before saving.
  - Store dynamically generated responses for popular queries.
  - How to maintain consistency?
    - kill.

- Fast Request & Processing
  - Store URL to file name mapping.
  - Catalogue access and basic file attributes (size, last-update time) which are to be used in response.
  - Response Header information (such as type) stays unchanged and can be pre-catalogued.

Server Performance:
Cookies & State Management

Client ➔ Origin Server A ➔ Request
Client ➔ Origin Server A ➔ Response
Client ➔ Origin Server A ➔ Set-Cookie: XYZ
Client ➔ Origin Server A ➔ Requests
Client ➔ Origin Server A ➔ Cookie: XYZ
Client ➔ Origin Server A ➔ Origin Server A

Server A sends Set-Cookie response in response headers to client C. Then all subsequent requests from client C to URL A contains the cookie.

Privacy Concern from Cookies

- Web-sites can identify users and start accumulating session history.
- 3rd party sites can also collect information about used by embedded URLs (Web Bug).

- User Controls:
  - Decide if one will accept it or not.
  - Set limit on size and number.
  - Set acceptance based on site/site/domain.
  - Set limit on the duration.
  - Set limit on current page sites.

Server Farm

- Multiple Web site on a Single Machine
  - Separate content authoring from hosting.
  - Advanced computing and networking infrastructure.
  - Cost amortization by mixing services.
  - Privacy of the content author.

- Concerns:
  - IP address depletion.
  - Synchronization for content upload.
  - Virtual hosting.
Virtual Hosting

- HTTP/1.0 browsers. To get bar.html from www.fox.com may send a request “GET that.html HTTP/1.0” to www.fox.com.
- However, if www.virtualhost.com does not see the vanity URL, then it cannot get bar.html to the intended server. Thus HTTP/1.0 requires distinct IP for each Web server.
- Solution: HTTP/1.1 introduced the “Host” header, which can specify the vanity URL.
  - GET that.html HTTP/1.1
  - Host: www.fox.com
- The vanity URL can be resolved to www.virtualhost.com’s server IP. But the server by looking the vanity URL would need to redirect the message to appropriate server.

Server Replication

- Mirror Site:
  - Replicate the content in multiple servers to enable load distribution.
  - How to update and keep them synchronized?
  - How to redirect requests?
- Server Bank:
  - Keep all in one logical front-end server. Locally distribute the load to individual servers working in the backend.
  - How to distribute request sequences evenly?
  - Replica file system update.
  - Access control/cookie management.

Server Bank

- Serving high rate of requests requires multiple web-server machines to work concurrently.

Case Study: A Business Site

Reference: Aden, Krishnamurthy and Reba, Hewlett-Packard Laboratories, Aug 2001

Workload Characterization

- (Resource Types)

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>March 2000 (%)</th>
<th>July 2000 (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic</td>
<td>96.58</td>
<td>96.73</td>
</tr>
<tr>
<td>HTML</td>
<td>2.58</td>
<td>2.67</td>
</tr>
<tr>
<td>Image</td>
<td>0.37</td>
<td>0.16</td>
</tr>
<tr>
<td>Text</td>
<td>0.03</td>
<td>0.01</td>
</tr>
<tr>
<td>Other</td>
<td>0.44</td>
<td>0.63</td>
</tr>
<tr>
<td>Total</td>
<td>100.00</td>
<td>100.00</td>
</tr>
</tbody>
</table>

In Effie, all resources containing a non-numeric extension, such as “examplexyz.html”, less than 1% in the dynamic category. They can only be generated by application servers, not web servers. And, of course, not all data generated by app servers is dynamic.

Workload Characterization

- (Usage Analysis)

Fig. 2. Traffic volume: (a) March 2000; (b) July 2000.
### Classes of Requests

<table>
<thead>
<tr>
<th>Types</th>
<th>Cacheable</th>
<th>Noncacheable</th>
<th>Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examples</td>
<td>Product descriptions</td>
<td>Photos</td>
<td>Adding item to shopping cart</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S or D?</th>
<th>Static</th>
<th>Dynamic</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personality</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q</th>
<th>1</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cacheable</td>
<td>77%</td>
<td>91%</td>
<td>92%</td>
</tr>
<tr>
<td>Noncacheable</td>
<td>23%</td>
<td>9%</td>
<td>8%</td>
</tr>
<tr>
<td>Search</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

### Session Analysis

- Every session is assigned a session number.
- A single session times out when over 15 mins.
- Robots consume resources, which we don’t want to see, but it’s difficult to tell the difference between a human user behavior and a robot behavior.
- In order to identify a robot in our survey, we assume sessions consisting of more than 30 requests are from robots. A detail analysis found that 9% of those sessions characterized as robots performed checkout operations.

### Table II. Relative Mean CPU Demand for Request Classes

<table>
<thead>
<tr>
<th>Request Class Name</th>
<th>Ratio of CPU Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cacheable (response can be cached)</td>
<td>1 (every 3rd)</td>
</tr>
<tr>
<td>Noncacheable (response cannot be cached)</td>
<td>100</td>
</tr>
</tbody>
</table>

### Session Analysis (Classes of Sessions)

- Fig. 2. Clustering based on request cache hit for users, July data ($d_t, d_{th} = 4$ is 1.47; $d_t, d_{th} = 7$ is 1.12).
- Over 60% of user sessions are characterized as hardly cacheable. 96% (99%) requests by these users are cacheable, which means they seldom do searching or checkout. They are “window shoppers.”

### Next Class...

Cacheing...