Processes
Processes

- Illusion of a Virtual processor
- Process state
  - CPU context (registers, PC, stack pointer, etc)
  - MMU and TLB
- Thread usage in non-distributed systems
  - Continue execution even after making blocking system calls
  - Increase parallelism in multiprocessor systems
  - Simpler to develop large applications
  - Inter Process Communication (IPC) is expensive
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

S3: Switch from kernel space to user

Operating system

Process A

Process B
Thread implementation

- User-level threads
  - + Easy to create/destroy
  - + Cheap to switch context between threads
  - - Invoking a blocking system call blocks all the threads

- Kernel –level threads
  - - Create/destroy/switching is expensive

- Kernel + User level threads
  - Light Weight Process (LWP) runs within the context of a process in Kernel space
  - Multiple LWPs per process
  - Create/destroy/synch threads in user space
  - LWP calls the user space thread scheduler
  - User space threads call thread scheduler as well
  - a system call is performed by switching to LWP in kernel mode
Thread Implementation

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

Scheduler activations by an upcall from Kernel to thread package
Threads in Distributed Systems

- Multithreaded clients
  - Web browsers: spawning multiple threads to download web page fragments (possible from multiple servers)

- Multithreaded Servers
  - Dispatche thread
  - Worker threads
Multithreaded Servers

- A multithreaded server organized in a dispatcher/worker model.
Multithreaded Servers

Three ways to construct a server.

<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 X-Window System

The basic organization of the X Window System

Server machine

Application

Xlib interface

X protocol

Terminal (includes display, keyboard, mouse, etc.)

Client machine

X kernel

Device drivers
Clients

Thin clients for networking computing

- Desiderata: Run X-window applications over a WAN and small devices
  - Problem: high network latencies
- Re-engineer the X protocol; Compress X messages via a delta-coding+cache
- Application controls the pixels of the display directly, transfers updates using compression

THINC

- provide higher level display commands that a simple display server can execute
- Intercept application requests and translate to sequence of display server commands
- Queue display server commands and transfer in batches
Client-side distribution transparency

- **Access transparency**
  - Through client stubs generated by Interface Definition Language (IDL)

- **Location, migration, relocation transparency** via using appropriate naming systems

- **Replication transparency**: a possible approach

![Diagram showing replication transparency]

- Proxy replicates invocation request
- All replicas see the same invocation
Servers

Iterative vs concurrent servers

How do clients know the endpoint of a server?

a) Client-to-server binding using a daemon as in DCE
b) Client-to-server binding using a superserver as the inetd in UNIX
Servers

How to interrupt a server?
- Abort client (most common approach used!)
- Out-of-band data by using TCP’s ability to transmit urgent data

Statefull or state-less server?
- Stateless server: keeps no information about its clients and can change state without having to inform any client
- Soft-state server: keeps client information for limited time, then it behaves as stateless
- Stateful server: maintains persistent information about the clients
- Permanent state vs (temp) session state
  - 3-tiered applications
- Cookies
Server Clusters

- Transport-layer switches (TCP handoff)
  - Content-aware request distribution
- Distributed servers
  - Home network
  - Home agent
  - Home address (HA)
  - Care-of address (CA)
  - Route optimization in MIPv6
    - Store (HA,CA) pairs at clients
    - Different clients can get different CA
Code Migration

- Why migration?
  - Performance, maintenance
- Each process has
  - Code segment
  - Resource segment
  - Execution segment
- Weak mobility
  - Transfer only the code segment
- Strong mobility
  - Can transfer the code and the execution segment
- Mobility can be
  - Sender initiated
  - Receiver initiated
- Execution of migrated code can be at the target process or in separate process
Code Migration

code migration and its resource segment

Process-to-resource binding
- Binding by identifier: process refers to a resource by its identifier (e.g., url, port number, etc)
- Binding by value: process can use any resource that provides the same value (library, file, etc)
- Binding by type: process can use any resource of the same type (printer, etc)

Resource-to-machine binding
- Unattached resource: can easily move between machines
- Fastened resource: possible to move resource at high cost
- Fixed resource: resource can not move
# Code migration

Possible actions in code migration with respect to the resource segment

<table>
<thead>
<tr>
<th>process-to-resource-binding</th>
<th>Resource-to-machine binding</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>unattached</td>
</tr>
<tr>
<td>by identifier</td>
<td>MV (or GR)</td>
</tr>
<tr>
<td></td>
<td>fastened</td>
</tr>
<tr>
<td>by value</td>
<td>GR (or MV)</td>
</tr>
<tr>
<td></td>
<td>fixed</td>
</tr>
<tr>
<td>by type</td>
<td>GR</td>
</tr>
<tr>
<td>RB (or MV, GR)</td>
<td>RB (or GR, CP)</td>
</tr>
<tr>
<td>RB (or CP)</td>
<td>RB (or CP)</td>
</tr>
</tbody>
</table>

GR: establish global identifier
MV: move resource
CP: copy resource value
RB: rebind process to locally-available resource
Code migration in heterogeneous systems

- Late 1970s: PASCAL portability through machine-independent intermediate code
- Mid 1990s: portability of scripting languages and Java
  - Deja vous PASCAL
- Migrating virtual memory pages of whole virtual machine
  - Push-all and resend changes
  - Freeze-and-transfer all
  - Pull on demand