Processes

Overview

- Threads vs Processes
- Clients
- Servers
- Virtualization
- Code Migration

Threads versus Processes

Threads:

- Improve application responsiveness by allowing them to not block

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

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

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

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

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:

Thread Implementations

- User-space threads:
  - Creating/destroying threads is inexpensive

Thread Implementations

- User-space threads:
  - Creating/destroying threads is inexpensive
  - Switching context is very fast

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...

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

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:

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

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.

Servers

- A server is a process implementing a specific service on behalf of a collection of clients

Servers

- 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*.

Servers

- 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.

Model	**Characteristics**
Single-threaded	No parallelism, blocking system calls
Multi-threaded	Parallelism, blocking system calls
Finite state machine	Parallelism, nonblocking system calls

Design Issues for Servers (2)

- How does a client find a server? Need to know the end-point or port and the host address.

Design Issues for Servers (2)

- 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.

Design Issues for Servers (2)

- 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 server.

Design Issues for Servers (2)

- 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 server.
  - Using a superserver that selects on multiple ports and forks off the appropriate server when a request comes in.