Approaches to Server Design

- Single process serialized server
- Forking
- Process Mob
- Multithreaded Single Process
- Multithreaded Multiprocess
- Single Process Asynchronous I/O
- Mixed Asynchronous I/O with Threads
Single Process Serialized Server

- Process an entire request before servicing another request
- Easy to design
- Atomicity easy to achieve
- Potentially bad performance under load
Forking

• Master process gets request - spawns child process - in Unix: fork() + exec()
• Child process processes complete request then terminates.
• Achieves parallel servicing
• Avoids memory leaks - HOW?
• Not very efficient - WHY?
Fork

rc=fork();

– duplicate process; return 0 to child, child PID to parent.
Process Mob

- Single queue, multi-server model
- `fork()` a set of processes at startup
- Pass requests to available child process
- Child processes remain active
- Memory leaks now possible? How to avoid?
Multithreaded Single Process

- Use a thread rather than a child process
- Faster context switch than processes
- Consider user space threads vs. kernel threads vs. Process when designing!
- Threads can come and go, or be persistent
Single Process, Asynchronous I/O

• In theory, we want to minimize context switching. HOW?

• Most of the time a process waits for slow (compared to compute speeds) I/O to complete

• Idea is to use “waiting time” to do computation

• Check “flags” to know when an I/O has completed

• Requires control block structures to keep track of requests

• Hard to do unless OS environment supports it: e.g., UNIX select()
Mixed Asynchronous I/O with Threads

• Use one thread to do I/O asynchronously
• Use threads to process computations.
• Neatly partitions I/O from application logic
• This approach is used by Netscape Enterprise Server.
Stateless vs. Stateful Servers

- **Stateful**
  - server remembers something about client transactions (control block)
  - improves efficiency by reducing transport message size

- **Stateless**
  - server remembers nothing about client transactions
  - useful when transport protocols may unreliable
  - servers recover fast from crashes
Precise description of stateful server challenge

• Idempotent
  – An operation which can be applied multiple times and still produce the same result
  – Formally, operation O is idempotent iff
    » O( a ) = O( O( a ) )

• Example
  – Idempotent: x = z
  – Not idempotent: x = x +z

• In a stateful server
  – Request operations must be idempotent, or
  – The server must recover from failures
Comer advice

- If the network is unreliable or machines can crash then the server should be stateless
- Do you agree?