Server Software Design

- General Architecture: bind, listen and infinite loop
- Design Options:
  - Connectionless Vs. Connection-oriented
  - Concurrent Vs. Iterative
  - Concurrency Vs. Parallelism
  - Stateful Vs. Stateless Servers
- Design Issues
  - Reliability
  - Session-oriented
  - Simplicity
  - Multiple connections Vs. single connection (per socket)

Server Software Design (cont.)

- More Design Issues
  - Establishment overhead (e.g. HTTP)
  - Resources consumption (server may runs out of resources!); solution: polling
  - Developing an adaptive reliable transmission protocol is difficult: RTT estimation: accuracy, TCP or UDP
  - What if multicast is needed?
  - Robustness (server crashes)
  - Minimizing response time
- Server Types
Iterative Connection-oriented Algorithm

1. Create socket
2. Bind to port
3. Make passive! (listen)
4. Accept connections
5. Read, process and reply
6. Close

Issues:
- INADDR_ANY
- listen queue
- accept returns another fd

Iterative Connectionless Algorithm

1. Create socket
2. Bind to port
3. Read, process and reply
4. Close

Issues:
- uses unconnected socket:
  `sendto(s, reply, flags, to, tolen)`
  `recvfrom(s, req, len, flags, from, fromlen)`

Some UNIX System Calls .. Fork()

- Process Vs. program
- Process ID:
  `pid_t getpid(void)`
  `pid_t getppid(void)`
- Fork() system call
  `#include <sys/types.h>`
  `#include <unistd.h>`
  `pid_t fork(void); /* in child, and child ID in parent */`
- Wait() system call
  `pid_t wait(int *infloc); /* infloc indicates the termination status */`
**Fork()**

- Child inherits:
  - file descriptors
  - real user ID
  - current and root directory
  - signals masks
  - environment
  - resource limits
- Child does not inherit:
  - fork() returns
  - process ID (and parents process IDs)
- Fork Example

---

**UNIX Signals**

- UNIX IPC
- Signals Concept
  - Asynchronous events, interrupt.
  - E.g. <ctr>c (SIGTERM), DEL (SIGINT)
- More Examples
  - SIGCHLD, SIGPIPE, SIGUSR1
- What a process does with a signal
  - Do nothing (default is to terminate the process)
  - Ignore the signal (except SIGKILL, SIGSTOP)
  - Catch the signal

---

**UNIX Signals (cont.)**

```c
#include <signals.h>

void (*signal (int signo, void (*func)(int))(int);
/*returns a pointer to signal handler*/
```

* Signals Applications
* SIGUSR Example