# ISA 563: Fundamentals of Systems Programming

Network Sockets

Feb. 19, 2013

# Sockets

- A form of inter-process communication, like:
  - Pipes
  - FIFOs
  - Shared memory

  (We will discuss these and advanced socket concepts later)
- Sockets allow IPC within the same host, as well as on different hosts on the network

# Socket Operations

Create an endpoint of communication:

```
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

| domain: | |
|---|---|
| AF_INET | IPv4 Internet domain |
| AF_INET6 | IPv6 Internet domain |
| AF_UNIX | UNIX domain |

| type: | |
|---|---|
| SOCK_DGRAM | Datagram |
| SOCK_STREAM | Connection |
| SOCK_RAW | Raw socket |

Close communication endpoint:

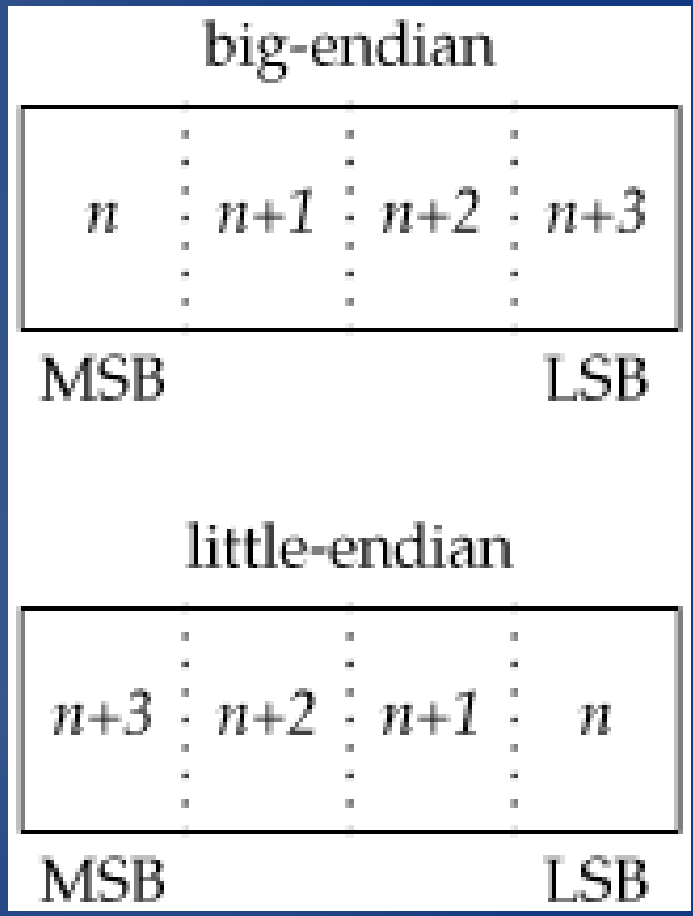
```
#include <sys/socket.h>

int shutdown (int sockfd, int how);
```

# Socket Operatoins (cont'd)

- The socket() function returns a file descriptor
  - The socket file descriptor can be treated as any other file descriptor from open().
    - close
    - read
    - write
    - ...
  - However, some functions cannot be used:
    - lseek
    - ftruncate
    - ioctl depends on driver
    - ...

# Byte order

- Endianness due to CPU architecture:
  - Little-endian: Intel
  - Big-endian: SPARC, PowerPC
- Has to have a common byte order for host-to-host communication:
  - Network byte order: big-endian



# Socket Operations: Client-side

- Create a socket:
  - `int sock = socket ( AF_INET, SOCK_STREAM, 0 ) ;`
- Connect to destination host and port:
  - `int val = connect ( sock, ... ) ;`
- Data exchange:
  - `int r = read(sock, buf, sizeof(buf));`
  - `int w = write(sock, buf, n);`
- Close connection:
  - `close(sock);`

# Demo

www-client.c

# Socket operations: Server-side

- Create a socket:
  - `int sock = socket ( AF_INET, SOCK_STREAM, 0 ) ;`
- Bind to port
  - `int val = bind ( sock, ... ) ;`
- Listen (for connection oriented protocols):
  - `int val = listen(sock, QLEN);`
- Wait for incoming connections:
  - 
```
for ( ; ; ) {
    conn = accept ( sock, ...) ;
    // process requests using conn ...
}
```

# Demo

echo-server.c

# Handling Simultaneous Requests

- Different ways of achieving concurrency:
  - fork
    - Parent spawns a separate process
      - Multiple process
  - select
    - Parent listens to multiple requests concurrently
      - Single process
  - threads
    - Parent creates a thread to process each request
      - Single process, multiple threads

# Concurrency through forking

- Parent:
  - Spawn of a copy of current process – fork()
- Child:
  - Closes listening socket
  - Processes incoming request
  - Closes client socket
  - Exits

# Demo

concurrent-echo-server.c