Overview

- *Administrative*
  * HW# 5 Due next week
  * HW# 5 : Any Questions
- *Topics*
  * Client Server Communication
  * 12.3 ISO/OSI Layers
  * 12.4 UICI Implementations
  * App. B (UICI : Socket Implementation)
  * 12.4 Sockets
  * 12.8 UICI and thread safety
- *Readings:*
  * 12.4 UICI Implementations
  * App. B (UICI : Socket Implementation)
  * 12.8 UICI and thread safety

12.3 ISO/OSI Layers

- *Idea*
  - Network protocols are complex
  - Must deal with many types of errors and services
  - Decompose as a set of layers
  - Lower layers provide services to upper layers
  - Lower layers hide details, reduce complexity
- *Layers - See Figure in the textbook*
  - Physical
  - Data link
  - Network
  - Transport

12.4 Sockets

- *History*
  * early-1980s: BSD 4.1 introduced it
  * mid-1980s: BSD 4.2 standardized it
  * Available in most Unix & Windows systems
  - linking: -lsocket, -lnsl
  * Part of Spec 1170
  * But not standardized by POSIX yet
- *Sockets vs. File Descriptors*
  * read/write operations are similar
  * creation is more complex
  - open() for file descriptor
- *Sockets vs. UICI Concepts*
  * Sockets = port
  * a handle to network communication channel

12.4 Sockets vs. UICI

- *Communication-style*
  - UICI: Connection-based
  - Sockets: Connection-based or connection-less
- *Server Design (Section 12.1)*
- UICI: Server design 3, 4, 5
- Socket: designs 1, 2, 3, 4, 5
- *Blocking*
  - UICI: many system calls
  - Socket: fewer system calls are blocking
- *System calls (Table 12.2, pp. 444)*
  - read/write is simple in both
  - Connection setup more complex in sockets

12.4 Sockets - Client System Calls

- *UICI Client (Program 12.4, pp. 442-3)*

```
fd = u_connect(portnumber, server_hostname);
u_read(fd , ..., ...);
u_close(fd);
```

- *socket based UICI implementation*

  * See u_close (pp. 607)
  - calls close() on file descriptor

  * See u_read (pp. 607)
  - calls read() on file descriptor
  - checks for signal interuptts

- *See u_connect (Program 12.7, pp. 452-3 or 606-7)*

```
sock = socket(AF_INET, SOCK_STREAM, 0);
/* initialize struct server */
connect(sock, &server, sizeof(server));
return sock;
```

12.4 Sockets - Basic System Calls

- *System call socket() - Synopsis (pp. 448)*

  int socket(int af, int type, int protocol)

  * Purpose: Create a socket, i.e. a port
  * Return value: file descriptor to new socket
  * Argument 1: address family or domain
  - AF_INET for network communication
  - AF_UNIX for single unix system to mimic pipe/FIFO
  * Argument 2: communication-type
  - SOCK_STREAM for connection based
  - SOCK_DGRAM for connection less
  * Argument 3: protocol
  - value 0 => system will choose appropriate one

- *Example Usage*

  * Example 12.5 (pp. 448)

12.4 Sockets - Basic System Calls

- *System call connect()*

  int connect(int s, struct sockaddr *name, int namelen)

  * Purpose: form a network communication channel
  - associate client port with a server port
  * Argument 1: file descriptor for client's socket
  * Argument 2: server's address (host, port)
  * Argument 3: size of 2nd argument
  * Returns 0 on success, -1 on error

- *Server address*

  * struct sockaddr_in (pp. 449)
  * sin_family - AF_INET
  * sin_port - port/socket number
  * sin_addr - hostname, e.g. deca.cs.umn.edu
  * sin_zero - filler for legacy compatibility

12.4 Sockets - Other System Calls

- *Other utility system calls*
  * gethostbyname() - synopsis on pp. 452
  * Argument: symbolic hostname
  * Returns: a structure
  - h_addr_list[] = network addresses used by this host
  - h_length = number of bytes in address
  * gethostbyname_r() - thread-safe version

12.4 Sockets - Byte Ordering

- *Network Byte Order for Numbers*
  - * little endian - bytes stored in increasing addresses
  - - Ex. Intel, Vax
  - * big endian
  - - Other architectures
- *Network Byte Order*
  - * Standard format, e.g. big-endian
- *Conversion Routines*

```
#include <sys/types.h>
#include <netinet/in.h>
u_long htonl(u_long hostlong);
u_short htons(u_short hostshort);
u_long ntohl(u_long netlong);
u_short ntohs(u_short netshort);
```

- *Implementation Details*
  - * These functions are macros
  - * little-endian machines - change bytes to network byte order.
  - * On big-endian machines no change needed

12.4 Sockets - Connection based Server

- *UICI Server (Program 12.2, pp. 439-440)*

  listenfd = u_open(portnumber);
  communfd = u_listen(listenfd, client);
  u_read(communfd , ..., ...);
  u_close(communfd);

- *socket based UICI implementation*

  * Already saw See u_close and u_read (pp. 607)

  * See u_open (Program 12.7, pp. 450)
  - uses socket(), bind(), listen()

  * See u_listen (Program 12.6, pp. 451)
  - uses accept()

- *Let us look at bind, listen, accept*

12.4 Sockets - Server System call

- *System call bind()*

  int bind(int s, const struct sockaddr *a, size_t a_len)

  * Purpose: Associate a socket file descriptor to
  - a network communication channel (e.g. port)
  * Argument1: file descriptor for socket
  * Argument2: address of server (hostname, port)
  * Argument3: length of argument 2
  * Returns 0 on success, -1 on error

- *System call listen()*

  int listen(int s, int backlog)

  * Purpose: Specify number of client request to be
  - buffered before server refuses a connection
  * Note: Quite different from u_listen()
  * Argument1: file descriptor for public socket
  * Argument2: buffer size
  * Returns 0 on success, -1 on error

- *Implmentation of u_open (Program 12.5, pp.450)*

12.4 Sockets - Server System call

- *System call accept()*

  int accept(int s, struct sockaddr *a, int *a_len)

  * Purpose: Server waits for client requests
  * Returns a file descriptor
  - for communication with the client
  * Argument1: file descriptor for socket
  * Argument2: address of client requesting connection
  - Filled by accept()
  * Argument3: length of argument 2

- *Implmentation of u_listen (Program 12.6, pp.451)*

  * Uses accept()
  * Uses gethostbyaddr() : Synopis pp. 451
  - to get symbolic name of host of client

12.4 Sockets - Client Protocol Summary

- *Summary of Client Side Protocol*
- *Steps on Client Side*

  * 1.Create a socket
  - with the socket() system call

  * 2.Connect the socket to the address of the server
  - using the connect() system call

  * 3.Send and receive data.
  - There are a number of ways to do this,
  - e.g. the read() and write() system calls.

12.4 Sockets - Server Protocol SUmmary

- *Summary of Server Side Protocol*
- *Steps on Server Side*
  * 1.Create a socket with the socket() system call
  * 2.Bind the socket to an address
  - using the bind() system call.
  - address = <port number, host machine>
  * 3.Specify buffer size for pending connections
  - with the listen() system call
  * 4.Accept a connection with the accept() system call.
  - typically blocks until a client connects with the server.
  * 5.Send and receive data

12.8 UICI - Thread Safety

- *Implementation of UICI has unsafe system calls*
  - gethostbyname() - pp. 452
  - gethostbyname_r() - thread-safe version
- *Thread safe UICI*
  - u_open_r
  - u_close_r
  - u_connect_r
  - u_listen_r
  - u_read_r
  - u_write_r