Overview

- *Administrative*
  * MQ solutions on the web
  * Grades so far
- *Topics:*
  * What is Client-Server Architecture?
  * 12.1 Server Architecture - 5 design patterns
  * 12.2 Universal Internet Comunication Interface (UICI)
  * 12.3 Network Communications (ISO/OSI Layers)
  * Sockets
  * Revisit UICI
  - 12.4 Socket implementation
  - 12.8 Thread safety
- *Readings: Chapter 12 (pp. 429-476)*
- *Exercises: client-server 1, 2, 3, 9, 10.*

What is Client-Server Architecture?

- *Motivation for Client-Server Architecture*
  - * Distributed Computing, e.g. intranet, internet
  - * Simplifying life for user
  - - e.g. Get my email from yahoo, hotmail, cs.umn.edu
  - - e.g. Access my files on any file server
  - * Simplifying life for OS & businesses
  - - Multiple provider for a well defined service
- *Concepts in Client-Server Architecture*
  - * Server processes
  - * Client processes
  - * Communication mechanism
  - * Naming mechanism

What is Client-Server Architecture?

- *Analogy with a common business model*
  * Servers: shops in a (virtual) mall!
  * Clients: customers
  * Communication:
  - Connection based: in person, telephone, web (tcp, ftp)
  - Connection less: snail mail, email, web (ip, http),
  * Naming: store address/id, credit card, <bank, check#>, ...
- *Protocol*
  * Client makes a request for a service to a server
  * Server provides the service to client
  * Server may be on same machine or a different machine

What is Client-Server Architecture?

- *Server processes*
  - * Provide services to client on a network
  - * Usually run in infinite loop as daemons
  - - waiting for requests from clients
  - * Ex. file server - manage disk, backup, file sharing
  - * Ex. mail server, file transfer servers, etc.
  - * Web: search engines, AOL, hotmail.com, realaudio, ...
- *Client processes*
  - * Processes using services from servers
  - * Send requests to servers, monitor status, etc.
  - * May be short lived
  - * May interact with multiple servers
  - * Ex. shell, IE/NS browser, etc.

What is Client-Server Architecture?

- *Communication mechanism*
  * OS support to convey requests, results
  * Ex. pipes, signals, files, etc.
  * Q? Will these work across internet of machines ?
  * No, these assume a common operating system!
  * New mechanism (Focus of Ch. 12)
  - Ex. sockets, UICI, ... (network of OS)
- *Communication mechanism*
  * Connection-less protocols
  * Connection-oriented protocols
- *Connection-oriented protocols are*
  * Robust - handle some network errors
  * But slower due to error management

What is Client-Server Architecture?

- *Connection-less Communication Protocols (e.g. email)*

  * Setup
  - Server publishes its address and service
  - Server waits for service requests

  * Service requests
  - Client sends a service request to server address
  - Server performs service and returns a reply

- *Connection-oriented Communication Protocols (e.g. telnet, ftp)*

  * Setup:
  - Server waits for a connection request from client
  - Client requests connection
  - connection established b/w client and server

  * Service requests
  - via a handle (e.g. file descriptor)
  - multiple requests are possible on the connection
  - server address not needed after setup

What is Client-Server Architecture?

- *Naming mechanism for processes*
  * Q? How did we uniquely identify processes so far?
  * Q? Will it work across internet of computers?
  * Problems of process-id
  - not unique across different OS
  - not unique across 2 invocations of same program
- *New mechanism : <Host, Port>*
  * Ex. www.cs.umn.edu:80
  * Host = unique name for the machine hosting server
  - Ex. symbolic name, e.g. deca.cs.umn.edu
  - or IP address, e.g. 160.94.120.47
  * Port = integer name for a mailbox on a host
  - Unique stable number assigned to each service
  - Server listens to assigned port
  - echo (7), daytime(13), ftp (21), telnet (23),
  - See file /etc/inet/services for other ports

12.1 Server Architecture - 5 design patterns

- *Server Design Options*
  - * Options for Number of incoming ports
  - - only 1
  - - many 1 per client
  - * Options for Number of outgoing ports
  - - only 1
  - - many 1 per client
  - * Process structure Options
  - - One Process, One thread
  - - Many Processes (1 process per client)
  - - One Process, Many thread (1 thread per client)
- *Comparison criteria*
  - * Security/Privacy for messages to client
  - * Long request slowing down small requests
  - - port level
  - - server level

12.1 Server Architecture - Design 1, 2

- *Server Design 1*
  - Number of incoming ports = 1
  - Number of outgoing ports = 1
  - Process structure = 1 process w/ 1 thread
  - See Figure 12.1, pp. 433
- *Server Design 2*
  - Number of incoming ports = 1
  - Number of outgoing ports = many (1 per client)
  - Process structure = 1 process w/ 1 thread
  - See Figure 12.2, pp. 434

12.1 Server Architecture - Design 3

- *Server Design 3*
  * incoming ports = 1 common + (1 per client)
  * outgoing ports = many (1 per client)
  * Process structure = 1 process w/ 1 thread
  * See Figure 12.3, pp. 434
- *Q? Compare the three designs for*
  * Security/Privacy for messages to client
  * Long request slowing down small requests
  - Assume server psuedocode like (Example 12.1, pp. 435)

12.1 Server Architecture - Design 4, 5

- *Server Design 4*
  * incoming ports = 1 common + (1 per client)
  * outgoing ports = many (1 per client)
  * Process structure = many (1 process per client)
  * See Figure 12.4, pp. 435
  * See Psuedo-code in Example 12.2 (pp. 435)
- *Server Design 5*
  * incoming ports = 1 common + (1 per client)
  * outgoing ports = many (1 per client)
  * Process structure = 1 process w/ many threads
  - 1 thread per client
  * See Figure 12.5, pp. 436
- *Both provide*
  * Private channels to each client
  * Long request won’t slow down small requests
  * Design 5 has lower overhead

12.2 Universal Internet Comunication Interface (UICI)

- *Focus: Client-Server Communication*
- *Semantics*
  * Supports connection-oriented communication
  * Ex. design 3, 4 or 5
  * A common port for connection request
  * Private two-way channel to each client
  - for subsequent read/write
- *Convention*
  * Similar protocol as files (Chapter 3)
  - open, close, read, write, + few new calls
  * Return value convention:
  - Most calls return -1 for error
  - exception: u_error() returns void

12.2 UICI - system calls

- *Summary of Syntax*
  - Table 12.1 (pp. 437)
- *Open, close*

  int u_open(u_port port)
  - Open file descriptor bound to "port"
  - Returns listening file descriptor

    int u_close(fd)
  - Close the handle
- *Read, Write*

  ssize_t u_read(int fd, char *buf, size_t nbyte)

  ssize_t u_write(int fd, char *buf, size_t nbyte)
  - read/write "nbyte" from "buf" to/from "fd"
  - Return number of bytes actually read/written
- *Q? Compare four system calls with those on files (Ch. 3).*

12.2 UICI - system calls

- *Connection Setup: new calls*

  int u_listen(int fd, char *hostn);

  * Server listens to connection request on "fd"
  * system call returns a new communication file descriptor
  - Server will use this file descriptor to talk to client

  int u_connect(u_port_t port, char *the_host);

  * Client requests connection to server <the_host, port>
  * System call returns a new communication file descriptor
  - Client will use this file descriptor to talk to server

- *Other calls*

  void u_error(char *errmsg)

  * Outputs "errmsg" followed by a UICI error message

12.2.2 UICI - Client protocol

- *Client*
  * request connection to specific <host, port>
  * connection request returns communication handle
  * client reads/writes to handle
  * client closes the handle
- *UICI System call usage protocol*

  comm_fd = u_connect(portnumber, hostname)
  u_read/u_write(comm_fd, ...) /* request service */
  u_close(comm_fd) /* service request */
- *Program 12.4 (pp. 442-3)*
  * Notice protocol - UICI system call sequence
  * Client reads file from stdin
  - and transfers file to server
  * Program 12.1 (pp. 438-9) for server is complementary
  - reads file from network and write to stdout

12.2.1 UICI - Server protocol

- *Server*
  - * Listens to connection requests on a "port"
  - * Server may translate "port" to a file decriptor
  - * generates new handle for communication for each request
  - * server serve request
  - - by reads/writes to client comm. handle
- *UICI System call usage protocol*

```
listenfd = u_open(portnumber)
/* loop on requests */
        comm_fd = u_listen(listenfd, client)
        u_read/u_write(comm_fd, ...) /* service request */
        u_close(comm_fd) /* service request */
```

12.2.1 UICI - Server protocol

- *Serial server*
  * Program 12.2 (pp. 439-440)
  * Refers to Program 12.1 (pp. 438-439)
  * Check the protocol - sequence of system calls
- *Q1. Analyze Programs 12.2 (pp. 439-440) to answer the following:*
  * Does small request wait for large request to finish?
  * Does each client have a private channel?
  * Identify Server architecture (1, 2, 3, 4 or 5)
  * Identify communication (connection-less or connection-based)
  * Does it have busy wait?
  * Why does it close "listenfd" ?
  * What happens if "portnumber" (argv[1]) is not available?
  * What happens if we run out of file descriptors for u_listen ?
  * Q? What happens if network is not reliable?
- *Q? How will I run Programs 12.2 and 12.4 together?*

12.2.1 UICI - Server protocol

- *Analysis of Programs 12.2 (pp. 439-440)*
  * Does small request wait for large request to finish? YES.
  * Does each client have a private channel? YES.
  * Server architecture : 3.
  * Communication is connection-based
  * Does it have busy wait? NO assuming u_listen blocks.
  * Why does it not close "listenfd" ?
  - Should u_close(listenfd) before exit(0);
  - Assumes OS will recycle listenfd resources
  * What happens if "portnumber" (argv[1]) is not available?
  - server report error details and exits
  * What happens if we run out of file descriptors for u_listen ?
  - server exits without reporting error details
  * Q? What happens if network is not reliable?
  - No effect, connection oriented comm. recovers from error.
- *Q? How will I run Programs 12.2 and 12.4 together?*
  * See Exercise 12.1 (pp. 444)

Exercises on UICI Servers

- *Non-serial server*
  * Program 12.3 (pp. 440-442)
- *Q1. Analyze Programs 12.3 to answer the following:*
  * Does small request wait for large request to finish?
  * Does each client have a private channel?
  * Identify Server architecture (1, 2, 3, 4 or 5)
  * Identify communication (connection-less or connection-based)
  * Does it have busy wait?
  * Why does it close "listenfd" ?
  * What happens if "portnumber" (argv[1]) is not available?
  * What happens if we run out of file descriptors for u_listen ?
  * What happens if we run out of processes on OS ?
  * How many processes and threads are in the server?
  * Can there be orphan processes?

12.2.3 UICI Implementations

- *Implementation Choices*
  - Many network protocols
  - Examples: sockets, TLI, STREAMS
  - Implementation sketched in Table 12.2 (pp. 444)
  - Note UICI is simplest, i.e. fewest system calls
- *Implementation Mechanisms offer capabilities beyond UICI*
  - Connection-less communication
  - non-blocking I/O, e.g. read()
- *Implementation Issues*
  - Q? What happens if network is not reliable?
  - Q? What happens to client if server dies?
  - Q? What happens to server resources if client dies?