CS4516 Advanced Computer Networks **D12**

Program 1 **{March 28, 2012}** **70 points**

Medical Examiner Client/Server
Using Go Back N in the Data Link Layer
Due: Friday, April 6, 2011 at 10 a.m.

This assignment consists of two components: a design paper and an implementation that supports multiple clients and a server for a Disaster Identification (DID) database. This program is to be completed in three-person teams. Each team must reserve a one-hour live demonstration of your project in an available slot on April 6, 9 or 10th.

The general assignment is to build a **concurrent server** that handles requests from two or more clients. **[*Linux socket calls are used with TCP to emulate a physical layer for transmitting between clients and a concurrent server*].** Both the clients and the server will have a small application layer protocol that defines the interaction between clients and the server.

When a new client connects to the concurrent server, the server will ***create*** a child to handle all interactions with that client. Whether the server children are ***forked as processes*** or generated using ***concurrent threads*** is a team design decision. However, forked processes will then require implementation of shared memory mechanisms to handle concurrent accesses to the database structures required in the design of your application.

Medical Examiner Disaster Identification Database

FEMA sees the need for a database and a client/server system to serve as a repository for information regarding bodies collected during a natural disaster. One function to be provided is the ability of authorized clients to input information such as photographs, fingerprints, dental records and tattoo information that could facilitate the identification of a body recovered after a natural disaster such as a flood, an earthquake or a tornado. The database includes information on where a body was discovered.

Other functions to be supported include: input by a qualified medical examiner that a body has been positively identified; clients with lesser access privileges (e.g., relatives from the general public) to issue queries to determine whether a family member's body has been positively identified and possibly a list of locations where unidentified bodies were found.

Medical Examiner/Family Member Application Layer

Assume FEMA has advertised for contract bids to implement a prototype for this network-based application. Your team must submit a design proposal for the Medical Examiner Client/Server system which includes a Disaster IDentification database.

The prototype DID database handles a maximum of 100 body entries and can be implemented using SQL or simply as an in-memory data structure that gets uploaded when the server starts. The medical examiner disaster application supports two classes of clients: **authorized clients** that have the

capability to input information about bodies recovered after a disaster that is sent to the DID database on the server and **query clients** that enable any member of the public to create an account and remotely query the DID database to retrieve current information about bodies recovered from a specific disaster and to indicate that they have a missing relative whose picture they may upload. You should use the following specifics to define the sizes of these **required** database fields:

id_number is a 9-digit identification number.
first_name is a non-blank ASCII string with maximum length of 15 characters.
last_name is a non-blank ASCII string with maximum length of 20 characters.
location is a ASCII string with maximum length of 36 chaaracters.

Your team will need other database fields depending on the functional specified in your design report.

Generic Application Layer Requirements

Given the time frame of this course, your design proposal needs to support only a few primitive operations in an application layer protocol that correspond to client/server interactions. However, the **minimum** requirement is at least **six** client request types that must include: inputting a photo of a person (or a tattoo) to the database,

Your application layer design needs to provide a clean mechanism for the client to notify the server when it is done sending a message (and visa-versa) and a scheme to close the client's connection to the server cleanly.

The Network Layer

The ***network layer*** receives messages from the application layer and creates packets consisting of: 256 bytes or less of message payload, two bytes for packet sequence number, and any other control bytes specified in your team's design proposal. The ***network layer*** sends packets to the ***data link layer*** and waits for packets received from the ***peer network layer*** through the ***data link layer.*** Received packets can be data packets or ACK packets or piggybacked packets. The ***network layer*** is also responsible for putting received packets together to construct messages to be sent up to the application layer. **Note – while the network layer and application layer are described separately here, it is likely they will implemented as a single layer.**

Initially, the ***client network layer*** makes a one-time call (note, this is an **emulation cheat**) to the physical layer to establish a connection with the ***server network layer***. Once a connection has been established, the client network layer begins receiving messages from the ***client application layer*** and depositing each message chunk into a packet payload.

Data Link Layer

The data link layer receives packets from the network layer, creates frames, and sends frames to the physical layer for transmission. The data link layer also receives transmitted frames from the physical layer, extracts the payload, reassembles packets, and forwards packets to the network layer.

Frame Format

All frames need a frame-type byte to distinguish data and ACK frames. All data frames must have two bytes for the sequence number, two bytes for error-detection, and one end-of-packet byte. The data link layer sends data frames that contain from 1 to 150 bytes of payload (encapsulated data from the network layer packet). ACK frames consist of **zero** bytes of payload, a two-byte sequence number, and a two-byte error detection field. Note - the setting of the end-of-packet byte indicates to the ***receiving data link layer*** that the current received frame is the last frame of a packet.

This assignment requires an implementation of the **Go Back N** sliding window protocol at the data link layer. Your design proposal may add other 'overhead' bytes to the frame structure deemed necessary to implement **Go Back N**. Your design needs to include two error-detection (FCS) bytes. Your design will need to include sequence numbers in the frames and a mechanism for handling ACK's (or NACK's if you choose this in your design)**.** The minimum frame size is your choice, and it is your design decision whether to ***piggyback*** ACK's or to send separate ACK frames. The data link layer continues to receive packets from its network layer until its sliding window is full of unACK'ed frames. This requires **multiple virtual timers** on both the client and server side.

Those teams seeking an A on the project and all BS/MS students must implement a sending window size of four frames. A sliding window of size one should be implemented and tested first. ALL teams struggling with this project can drop back and turn in a working version with a window size of one.

General Data Link Layer Issues

The **data link layer** has to check for transmission errors using the **error-detection** bytes. If the received data frame is in error, this event is recorded and the receiving process waits to receive another frame. While **CRC** at the bit level is possible, it is strongly recommended that you use a two-byte **XOR folding algorithm** of all the frame bytes to create your error-detection bytes and to validate frames upon reception. For the ACK frame, the error-detection bytes simply become a copy of the two-byte sequence number.

The Client Process Flow

Each Client will call the ***physical layer*** to establish a connection to the concurrent server. The ***data link layer*** then gets packets from the ***network layer***, puts together frames and sends them to the ***physical layer*** to transmit. The ***data link layer*** flow depends on events coming from the ***network layer***, the current availability within the sliding window, and events coming from the ***physical layer***.

The Server Child Process Flow

This assignment requires implementation of a concurrent server that supports concurrent conversations with multiple clients.

Each ***server data link layer process (or thread)*** waits for frames from the ***server physical layer*** and passes packets up to the ***server network layer***. Similar to the client side, the flow of the ***server data link layer*** depends on whether there is traffic from the Server to be sent back to the Client (either frames with packets from the ***server network layer*** or ACK frames that are part of the Go Back N protocol).

The Physical Layer

The ***physical layer*** uses Linux sockets to emulate the sending of constructed frames as actual TCP messages between the clients and the concurrent server. When the ***client physical layer*** first establishes a client connection to the concurrent server, it **must** send one TCP message to the server child process or thread to identify the client by name. While a client name from the server's perspective is not actually needed, for the purposes of the final demo it is quite useful to be able to tell the difference between 'Client A' and 'Client B' when something fails while two or more clients are running.

Frame Error Simulation

Since real TCP guarantees no errors at the emulated physical layer, your program must inject artificial transmission errors into your physical layer.

Force a **transmission error** in every 6^{th} data frame sent by flipping any single bit in the error-detection bytes prior to transmission of the frame. Force a **transmission error** in every 8^{th} ACK frame sent by using the same flipping mechanism. (i.e., data frames 6, 12, 18, … sent will be perceived as in error by the receiver and ACK frames 8, 16, 24, … sent will be perceived as error by the receiver.) Note: since both the clients and the server can send data frames and ACK frames each of these counts that trigger transmission errors must be kept relative to whichever node is sending data or ACK frames. When the ***client or server data link layer*** times out due to either type of transmission error, it retransmits the data frame with the correct error-detection byte such that the retransmission will not be received with an error.

Assignment Hints

- **[Design]** Plan your design in a modular fashion such that if everything is **not** totally working, you can turn in and demo some type of output that shows **exactly** what is working. Relaxing the sliding window scheme is one option when your project is in trouble.

- **[Documentation]** Your commented program **must** have a special section to explain the details of your specific design decisions. **Remember: This is a team project and all routines must specify the author as part of the documentation!! Team members may not receive the same grade on an assignment due to uneven workload.**

- **[DEBUG]** Include print statements in the various layers while debugging. You should use some type of verbose debugging flag that can be turned on and off.

- **[Performance Timing]** You **must** measure and print out the total execution time of the complete emulated transfer per client.

- **port numbers** - Your clients should have unique port numbers and the clients should treat the server port number like a "well-known" port number.

- **[Version retention]** Exercise extreme care at retaining old versions. Demo experience has shown that a project team should keep old versions (especially in the 11th hour at 3 a.m.) because **often** the newest version done in a rush yields significantly worse results and you need to be able to fall back to a previous version for the demo!

Deliverables

Each team will have to schedule a one hour demo. At the due date, each team should turn in a tarred/zipped file that includes the source code and a README file. At the demo, provide a short user guide that explains the client application commands and syntax.

The README file **must indicate how much of project is working at the time of the demo. It also needs to indicate the non-working parts and some indication of why specific modules are not working correctly. This README information is critical for teams to receive partial-credit for for non-operational versions of program 1.**

See the Design Proposal Report for more detailed information about what to turn in at the project demonstration.