April 29, 2016

Communication Module Design

Objectives

The objective of this project is to design a very simple communication model that emulates a real-time environment. The model is expected to be fully functional. You will be provided with the following input:

- file with sample raw data. ('com_mod_input.bin')

You may implement this in C++ using the STL or custom structures, and/or 3rd party code-bases (ex. Boost, Qt). The goal of this exercise is for you to demonstrate proficiency with networking, multi-threading, and object-oriented design. Your solution will be judged both for functioning correctly and for the quality of the code.

When you arrive for your second interview, you will be expected to present:

- The output of your solution, as well as to demonstrate how you tested your code.
- The architecture of your solution, including any format or protocol specifications.
- Code walk-through.
- Question and answer

Prior to the second interview, please send:

- Source code, results, compiled binaries OR build instructions
- A high-level documentation – architecture, design considerations, etc.
- Instructions for how to run the program

Platforms: The target platform may be either Windows or Linux.

Requirements

Please take into account the following requirements when designing the model:

- Create two modules (Module A and B) that communicate with each other over a socket. They represent front- and back-end modules in a single-to-many scheme, where there is only one instance of A, but potentially multiple instances of B

Module A

- Module A processes the raw data from the input file at a constant data rate (simulating a data stream from some hardware). The input file will represent a single cycle of a periodic signal, so when your program reaches the end of the file, it should go back to the beginning of the file and loop continuously, thereby simulating a physical source.
- According to the frame sync bit (see specification below), it will parcel out the input to frames, such that each asserted frame sync bit indicates the end of a frame
- For each channel, each frame will be processed to produce a single 32-bit checksum by simply summing all the double-words in the frame for that channel.
- Knowing the input data rate, Module A will calculate the checksum output rate and pair the output data with a time-stamp. The time-stamp will continuously increase as the input data loops. The time-stamp will be a double-precision floating value in units of 1 second.

Module B

- Module B is a client which connects to A via a socket and receives the checksums for all channels.
- The link to A will be one-way, with B receiving the data from A periodically
- Module B will then output its data to a file or to standard output. The format of the output is up to you, but you must specify it. It may be either binary or human-readable ascii.
- More than one instance of B can connect at a time. Module A should open a separate thread for each connecting client.
- Test that the output data does not contain any packet drops or frame shifts and that the per-timestamp output for any two B clients matches the output of A.

Implementation details

- **Input Data**

- Raw data is stored in the supplied file in binary format using the following structure:
  - Each packet consists of 8 data channels plus header. Each data sample is 4 byte long. The header is 4 bytes: bits [31:1] is the incoming packet number, bit [0] is the frame sync.

HEADER (32 bits)																
Packet Counter (31 bits)				Frame Sync (1 bit)	CH0 DATA (32 bits)				. . . . .				CH7 DATA (32 bits)			
B3	B2	B1	B0	LSB	B 3	B 2	B 1	B0	B 3	B 2	B 1	B 0	B 3	B 2	B 1	B0

- The file to be read in at the constant data rate of 1 packet per 15.625 microseconds.
- The Packet counter is just an incrementing number. Should be checked for consistency
- The Frame sync signal is LSB in the header. The rest of the bits in this group are unused.

Guiding questions for the presentation

- How did you test your implementation?
  - Tools, methods
- Discuss the end-to-end latency of your solution – what impacts it? What are some potential choke points?

OPTIONAL: Extended logic – high-level discussion in documentation

This part is meant as an extra on top of the project, provided you have the time and inclination - it is not mandatory.

Consider an expansion to B's logic, where it alerts for “suspicious checksums” from a certain channel. These are checksums that appear more than once – the more a checksum repeats, the more “suspicious” it is.

Assume the following:
- For two identically “suspicious” checksums, the one which was hit last is “more suspicious”.
- The problem is bounded: You may clamp per-checksum hit counts by M, and all checksums are within some range for the problem’s duration. However, you have no control over the range, and it may be very large.

Imagine the following pseudo-API in B:

unsigned int getMostSuspiciousChecksum() const;

Given that API, suggest (no coding) a solution to this problem. Discuss your suggestion in the context of space/time behavior and the option of multiple processors.