Implementation of a Linux C daemon to locally distribute AFDX-like avionics data

Piattaforme Software per la Rete
A.A. 2016/17
Student: Alberto Martin Lopez
Tutor: Federico Reghenzani

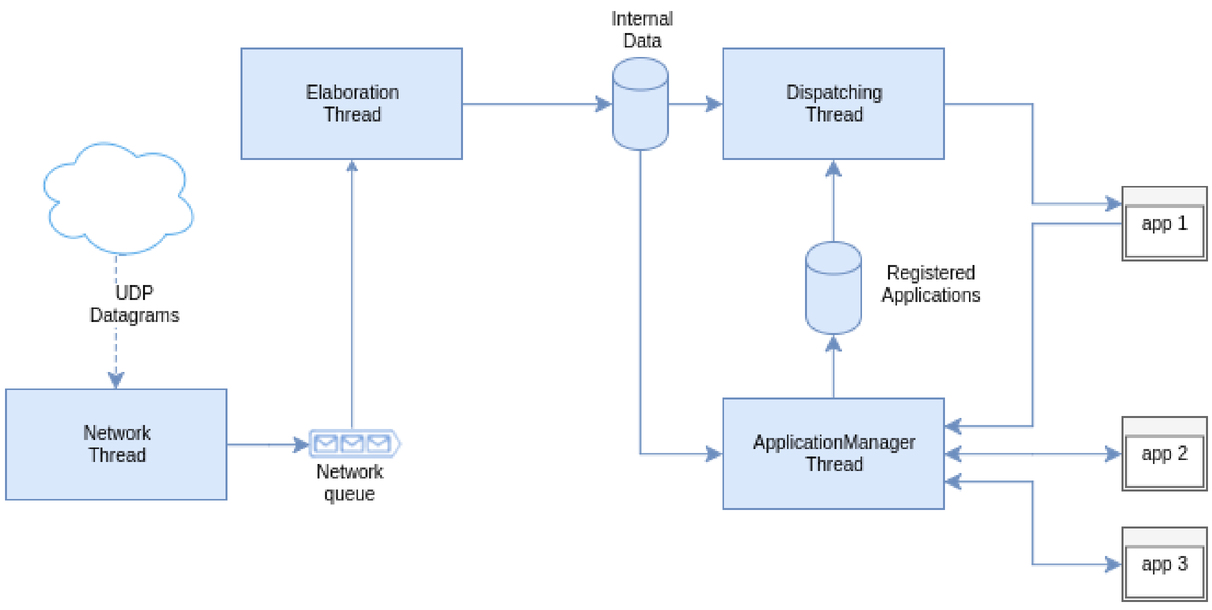
Background

The AFDX network is a protocol created to user the Ethernet stack in avionic applications. The major improvement in AFDX compared to Ethernet is the determinism guaranteed in timing and delivering of messages.

The idea of this project is to build a daemon that receives data from the AFDX network and propagates it to real-time applications. Obviously, for the project we don't have an AFDX network, so we will work with UDP over Ethernet. Furthermore, it's not required to fulfill any real-time constraints.

General idea

The general idea of the daemon is shown in the following picture:



The network thread catches the messages from the network (broadcast UDP datagrams), put them in a queue and return to wait new messages.

The elaboration thread performs some computation of the messages available in the network queue - removing them from the queue - and accordingly updates the internal data struct.

The applications may query the ApplicationManager via a named pipe (FIFO) asking directly a data or requesting a periodic update of a data. In the last case, the request is saved into a proper struct. The Dispatching Thread is in charge to periodically send the data according to the registered applications.

Receiving Data

The UDP datagrams are sent by the source as a broadcast message (check the SO_BROADCAST flag of the `setsockopt` function call). Each datagram is composed by an identifier and a payload. Please check the repository for the struct definitions.

Listening address and port should be selectable at command line:
e.g. *afdx 192.168.0.1 1235*

The internal queue must have a fixed size decided at compile time and can be implemented as you want. The receiver has to signal the elaboration thread when new data are available (hint: use semaphore).

Computation on received data

The elaboration thread acts as a “consumer” of the data produced by the network thread. The elaboration thread pops from the queue one packet at a time and based on its identifier, it performs some computations to the payload in order to update the internal structure.

I will give you in a second time what this thread has to perform, at present, just copy the data that do not require additional computations and discard the others (check the repository).

Obviously, pay attention to the race conditions on the shared data struct.

Application Manager

During startup it has to create a named pipe: “/tmp/afdx_daemon_fifo”
Any process may send a “welcome message” (check the repository), then the application manager creates the fifo “/tmp/afdx_{PID}_fifo” that will be used for the actual communication. The application manager has to maintain a list of opened fifo. The fifos have to be checked via the `select` system call, in order to leave the application manager to be single-thread.

Then the application can:

- ask one-shot for a data (the application manager will reply with the data)
- ask to be registered for a periodic data

I let you to decide how to implement the data structs for request/registration/sending data.