Tutorial ReadMe

Deepthi Gorthi

April 19, 2018

Hashpipe is a collection of useful structures and functions that will help you create a pipeline for high speed data collection from an ethernet port to say, a file system. This module is by no means a substitute to lack of knowledge about what goes into creating a high speed data pipeline or C programming. It is just a tool that will immensely aid your attempt if you already know what needs to be done.

Don't be discouraged though! It is easy to learn if your data rate is not very high (1Gbps at the time of writing).

Barebones Hashpipe Let's start on the axiom that you need a multi-threaded process. At the very least your pipeline should have a thread that takes data from a socket and feeds into a shared memory, another thread that reads this data, processes it and feeds into another shared memory and a final thread that writes it to disk. Depending on the datarate you are catering to, you can omit one or more of these steps, the limit being using a serial code that is not multi-threaded.

This tutorial contains just those three threads and a few additional ones to link up all of them; to perform the simple task of adding two numbers. Here is map:

1. `recv_data_thread.c`: In an actual application this thread takes in high speed data from a socket. For the purposes of this tutorial, it asks the user to input two numbers and stores it in a shared memory.

2. `process_data_thread.c`: This thread adds the two numbers and puts the sum in a second shared memory.

3. `output_data_thread.c`: This reads the second shared memory and print the sum to the terminal screen.

4. `databuf.c`: This program defines the size of the ring buffer you need for this process. The ring buffer (`hashpipe_databuf_t` structure) needs a header size (> 96 bytes), block size and number of blocks in the ring buffer.

   In principle, this file can be much more complicated with your custom structures defining the ordering of your data.

5. `databuf.h`: Header file containing the structures you want to define. In our case this just contains the function that creates our shared buffers.

6. `Makefile`: To link all the above threads, header files, hashpipe etc. and compile them. **NB**: Make sure you `make all` and `make install` for the plugin to be callable.

7. `init.sh`: Calls the plugin.