CMPUT 379 Assignment 2

Chengxuan Li

2022-02-02

1 Objectives

This programming assignment is intended to give you experience in developing communicating peer processes that utilize signals for examining the progress of the running processes, FIFOs for communication, and I/O multiplexing for nonblocking I/O.

2 Design Overview

- For both master switch and packet switches, they will automatically create all the required FIFOs for packet transfer if they don't exist in current directory. Regardless the type of switches, there will be always two FIFOs between two switches. For instance, there are two switches, switch A and switch B. There will be two FIFOs, one named "fifo-A-B", another named "fifo-B-A". "A-B" in "fifo-A-B" means the direction of packet go from switch A to switch B. This applies to "fifo-B-A" as well.
- For each packet switch, it will create two threads. One thread is for reading the ".dat" file, and another one is for I/O multiplexing that monitors the keyboard, the master switch, and the attached switches in a unblocking manner.
- The type of the packets and the content of the packet is stored in a struct called "Frame". The implementation is identical to the implementation provided in the lab experiments with sending and receiving formatted messages. The only difference is that the content of the packet struct consist of different members in integer type based on the type of packets instead of an array with fixed length. Each member in the content of the packet struct indicates the information required to send a packet for a specific packet type. For example, it requires the source IP and destination IP to send a ASK packet, which needs to fill in two members : int srcIP, and destIP.
- The program is divided into different modules, and is used object oriented programming. Information of master switch and packet switches are en-

capsulated class MasterSwitch and PacketSwitch repsectively. To access those information and operate the switches, it necessary to create an instance / object of a specific class and invoke the methods for a specific operation such as send ADD packet, reading ".dat" file and so on. Implementation of packets is in a separate module. Other implementations that are for functional purpose and error handling are also in a separate module.

3 Project Status

In the current status of the project, all the required specification specified in the assignment are implemented and tested. However, there are few exceptions due to difficulties encountered in the implementation although they have been solved by some work around solutions.

One is signal handling for signal "SIGUSR1". The signal handling function need to be static. To print out the information of a switch, it needs to invoke the a member method (non-static) called info() after creating a object of either class MasterSwitch or PacketSwitch. This member method cannot be passed into signal handling function. The work around solution used is to declare the object of class MasterSwitch (PacketSwitch) as global variable, and create a separate static function as signal handling function that calls info() from that object. A more preferable method (personally) that is not implemented is to use singleton design pattern along with using a static / class method.

Another difficulties is using thread to achieve non-blocking sleep so that a packet switch can still perform I/O multiplexing when it's delaying packet switch. One obvious issue of using thread is that the race condition between handling packet transfer and processing, which can cause some degree of desynchronization. There are some constraints installed in the program (mainly the file processing thread) to prevent potential desynchronization. The thread for processing ".dat" file will put into a while sleep loop until the packet switch and master finish the HELLO packet and HELLO_ACK transfer.

When the file processing thread encounters a line (let call it a sending request) that specifics a packet transfer from current packet switch to another, if there's no such a rule in the current packet switch, it will send ASK packet to master, and store the sending request in a pending queue. Also, this thread will update the sending history of ASK packet. This prevent it to send (received) duplicate ASK (ADD) packet. The file process thread will continue processing the data file without waiting ADD packet from master.

In the I/O multiplexing thread, it will not only monitor all the keyboard input and FIFOs, but also actively clean up the sending request in the pending queue if the pending queue is not empty at the beginning of polling loop.

4 Testing and Results

The test result of the program is identical to example output when it used ex1.dat. There were some differences in terms output order between test result and example output when it used ex2.dat and ex3.dat. The test results still matched the specification of the assignment. The differences were cause by the time gap between each switch open. Despite there are some constraints in place in the packet switch, threading is still the reason that produces the differences since the file processing thread and I/O multiplexing thread run in different pace. Particularly, it's possible for switch 1 to forward a packet to switch 2 if they open approximately the same time.

5 Acknowledgements

List of sources of assistance:

- Experiments with sending and receiving formatted messages
- APUE textbook
- How to pass class member function to pthread_create() ?