CMPUT 379 Assignment 2 Project Report

Name: Tymoore Jamal

Student ID: 1452978

LEC A1

Objectives:

This project had 3 main objectives. To familiarize us with client server relationships, to teach us how to send data through non-blocking inter-process communications, and to learn how to redirect signal usage in order to change the indented effect of a signal.

This provides us with meaningful experience interacting with the GNU operating system through making system calls to the Unix shell. This allows us to have significantly more freedom when programming in the Unix environment, as system calls are quite powerful and allow us to achieve results we would not otherwise be able to do.

Specific functions that this project focused on were pipe, sigaction, write, and read. The combination of these functions allowed us to create a software designed network with a controller and several switches, where all switches could read and write data to and from the controller and its neighboring switches in a non-blocking manner.

Design Overview:

The important features are outlined below:

a2sdn general:

- Upon execution set the CPU clock limit to be 10 minutes as a preventative measure, to ensure no process executes indefinitely.
- Determine if the user would like to execute a controller or switch

a2sdn controller:

- Upon controller creation, remove any pre-existing FIFOs from the current directory as we will be creating our own.
- Configure the process signal handler to route SIGUSR1 to a defined function which prints all needed values
- Create two FIFOs for each switch
  - One for reading
  - One for writing
- Construct the pollfd array
- Poll over all switches, and the keyboard
  - If list is provided from the keyboard list all data needed to be printed
  - If exit is provided from the keyboard, list all data needed to be printed and then end execution after shutting down
    - Shutting down entails deleting all FIFOs and returning the signal processing back to its default behaviour
- If the controller gets a message from the switch
  - Identify the type of message

        - If OPEN add the switches data to the controller’s memory and send ACK
        - If QUERY determine the rule to send back to the controller
            - DROP – If the destination IP is unreachable from that switch
            - FORWARD - If the destination IP is reachable from that switch
            - To determine if it is reachable and if so in what direction a recursive algorithm is used

a2sdn switch:

- Configure the process signal handler to route SIGUSR1 to a defined function which prints all needed values
- Create two FIFOs for each neighboring switch
    - One for reading
    - One for writing
- Construct the pollfd array
- Initialize the flow table
- Send OPEN to the controller
- Poll over all neighboring switches, the controller, and the keyboard
    - If list is provided from the keyboard list all data needed to be printed
    - If exit is provided from the keyboard, list all data needed to be printed and then end execution after shutting down
    - If we get a RELAY message from a switch we will attempt to process it, if we cannot then we QUERY the controller
    - If we get an ADD message from the controller we add a rule to our flow table (if it does not override another rule, we have)
- Process a line from the transfer file
    - If we find an instruction, we know how to process we do so immediately, this may involve sending a RELAY message to a neighboring switch, or dropping the rule
    - Otherwise we send a QUERY message to the controller to get an appropriate rule and we add the instruction to the waiting queue

Project Status:

a2sdn controller:

Some difficulties included:

- Keeping track of all file descriptors for all switches, and ensuring the writing FIFOs are not opened too early
- Implementing poll to have non-blocking I/O multiplexing
- Formatting receiving and transmitting messages properly
- Determining which rule to send back for a given IP address

a2sdn switches:

Some difficulties included:

- Keeping track of all file descriptors for all neighboring switches and the keyboard and controller
- Ensuring the writing FIFOs are not opened too early
- Managing the traffic file instructions
- Managing the switch flow table
- Managing and the instruction waiting queue and ensuring it's always valid
- Only querying the controller when necessary.
    - i.e. not sending the same query twice

Testing and Results:

I found that for this specific program testing has to be an ongoing and iterative process. This involved heavily testing each new aspect of the program as it was being developed.

E.g. first testing that a controller is created when the user wants a controller created and a switch is created when the user wants a switch created. Then moving on to communication testing in the simplest of cases, such as sending a short test string across a FIFO. From there I was able to then test more complex functionality as it was built, such as sending full object across the FIFO, handling multiple input FIFOs at the same time, managing the flow table and instruction queue.

Overall this approach was extremely beneficial for me as it allowed me to confirm that I was always on the right track and that the functionalities I implemented were working.

Acknowledgements:

I did this assignment individually and did not collaborate with anyone.

Assumptions Made:

- A user will provide a2sdn a valid set of inputs
- The file the user provides is of type .dat and follows the same format as the 3 examples posted
- There will be one and only one controller created, and it will be created before any switches are created
- Once a controller is created it will not be exited until the very end
    - i.e. no more switches are created after the controller exits
- All switches must have a valid number
    - A number between 1 and the max number of switches
- No two switches can have the same number
- Port 1 of Switch k will either be null or connect to switch k-1
- Port 2 of Switch k will either be null or connect to switch k+1
- Once a switch is created it will not be exited until the very end
    - i.e. no more switches are created after the switch exits
- Once a rule is inserted into the switch flow table another rule **CANNOT** override it
    - Any rules seeming to overwrite a previous rule will be not be added to the switch flow table.