Objective:

The objective of this project was to solidify a form of inter process communication through the use of named pipes and sockets. Furthermore, the implementation of non blocking I/O and I/O multiplexing emphasized communication from many processes to one process such that a process can run smoothly without blocking and only reads from those processes which have a message to communicate to the current running process.

Design overview:

Main:

- Takes in the command line arguments and checks for correctness of the arguments. If invalid arguments are specified then the program prints an error message and then quits.
- Calls the function init switch or init controller depending on whether the user specified a controller or a switch in the command line arguments

Init controller:

- Init controller receives the parameter which is number of switches and allocates space for an array of current switches structures which will contain the information on switches which send an “OPEN” Packet to the controller
- Creates a socket and stores the sockets file descriptor in an array of pollfd structs
- Initializes a struct containing the number of received and transmitted messages
- Calls the controller_loop function passing to it as parameters the data structures it initialized

Controller_loop:

- Polls the stdin for any input matching “list” or “exit”
- Binds to the port specified in the command line and listens to the number of switches also indicated in the command line arguments
- If a switch connects to the socket and sends an open then we add the switch to our array of structures containing our current switches. We add them with the information such as neighbours, low and high destination IP’s
- Once the switch had been added we send the switch back an ack packet
- If it was a query we call query_switches to get the current action to send to the switch which had queried
- Increments the counts of the received and transmitted messages

Init_switch:

- Extracts the information on left and right neighbours and destination IP and traffic file that were entered on the command line when the program was run
- Attempts to connect to the controller socket, if the connection fails then it sleeps for 2 seconds attempting a maximum of three times before the program will print an error message and then exit
- Sends a message to the controller of type “OPEN”
- Opens the file descriptors that other processes use to send messages to the current switch and store them in a pollfd struct array along with the socket file descriptor
- Initializes the switch stats structure which contains a count of all the received and transmitted messages
- Calls switch_loop function passing to it as parameters the data structures that it initialized

Switch_loop:

- Reads one line from the traffic file, if it does not find the matching rule in its flow table then it sends a message of type “QUERY” to the controller containing the destination IP from the line in the traffic file
    - If the line contains the word delay then we delay reading from the traffic file for the number of msec specified in the line
- Polls the file descriptors including the stdin
    - list information on switch or list then exit
    - relay any message from neighbour switches to the next neighbour if the message was not destined for the current switch
    - if information is received from the controller socket it is handled appropriately
        - ack
        - add: call execute cont rule to either forward or drop the packet

is_num:
- checks if the passed in parameter is a number

str_to_i:
- converts the passed in string into an integer

extract_sw_num:
- takes in a string usually of the form swk where k is 1…7 and extracts and returns the digit contained in the string

print_switch:
- prints the contents of the flow table
- prints the number of each type of received and transmitted message contained in switch_stats struct

print_controller:
- prints the pertinent information on the current connected switches from the struct current switches
- prints the number of each type of received and transmitted message contained in the cont_stats struct

update_flow_table:
- checks if the header is contained in the flow table
    - if not then it is added as a new element to the flow table
    - if so then it updates the counts

execute_cont_rule:
- takes in the message received from the controller and executes the action that it specifies including drop and forward.
- Calls update_flow_table to add the action contained in the message

Query_switches:
- Checks if there is a switch that the destination IP that has been queried to the controller exists
    - If it does not exists a message of type drop is created to send to the querying switch
    - If so a message of type forward is created to send to the querying switch

is_dig_char:
- Goes through an inputted string returning 1 if all the characters in the string are digit characters or a dot, otherwise it returns 0

Project Status:

The project right now is complete as it seems to be outputting the correct information and sending the correct messages. The implementation of the assignment was difficult especially researching how to properly use the socket functions and make sure that the socket of the switches were correctly connecting to the controller socket. The messages I had on assignment 2 were vague so it took some time to make sure that all the required transmitted and received messages were being printed to the stdout. Otherwise, it was straightforward to implement.

Testing and Results:

I had tested my implementation by entering invalid arguments on the command line and it could handle them correctly by printing an error message and then exiting the program. I had also tested for correctness by comparing my output to the given examples on the eclass page for two switches. There were many syntax type errors that were fixed until the output pretty much matched the output in the example. I had many output messages throughout the code to ensure that all the right messages were being passed and that each function was running to completion. I also tested with the three switches example from assignment 2 in order to ensure the new socket implementation was still working correctly for that case.

Acknowledgements:

- www.IBM .com
- advanced programming in the UNIX Environment 3rd edition
- man7.org/linux/man-pages
- Lecture slides CMPUT 379 by Ehab Elmallah
- www.stackoverflow.com
- www.geeksforgeeks.org