Objectives:
This programming assignment is intended to give experience in developing peer-to-peer programs that utilize signals for examining the progress of the running processes, FIFOs for communication, and I/O multiplexing for nonblocking I/O. In these terms, the assignment is not that usefull as most of the assignment is making the other features needed to perform the I/O moltiplexing. However, these concepts were all needed.

The program objective was to perform the transactions of a simple linear SDN.

How to use:
Makefile commands (assignment spec):
- Executing ‘make’ produces the a2sdn executable file.
- Executing ‘make clean’ removes unneeded files produced in compilation. However, no unneeded files are produced in compilation.
- Executing ‘make tar’ produces the ‘submit.tar’ archive.

Running the program:
The program can be invoked as a controller using
"a2sdn cont nSwitch"
where cont is a reserved word, and nSwitch specifies the number of switches in the network (at most MAX NSW= 7 switches).

The program can also be invoked as a switch using:
"a2sdn swi trafficFile [null|swj] [null|swk] IPlow-IPhigh"
In this form, the program simulates switch swi by processing traffic read from file trafficFile. Port 1 and port 2 of swi are connected to switches swj and swk, respectively. Either, or both, of these two switches may be null. Switch swi handles traffic from hosts in the IP range [IPlow-IPhigh]. Each IP address is ≤ MAXIP (= 1000) and >= 0. Sequence matters. The controller should be set up first and if switch i has switch j on a port, switch j must have switch i on a port.

Design Overview:
In general, c++ code is used and I made small functions that only do one thing to increase understandability of the functions. Code is also separated into a2sdn, connection, controller, switch, packet, and flowTable files.

1. a2sdn
a2sdn.cpp
- sets up SIGUSR1 signal

- parses command line argurments
- runs a switch or controller

2. switch
   switch.h
   - defines a switch
   switch.cpp
   - does all switch logic

3. Controller
   Controller.h
   - defines the controller class
   Controller.cpp
   - does all controller functionality and logic

4. connection
   connection.h
   - defines a port connection struct
   connection.cpp
   - handles fifo opening

5. packet
   packet.h
   - defines packets (switch rule deined separately)
   packet.cpp
   - gets and sends packets

6. flowTable.h
   - defines a switch rule

Project Status:
All functionality delivered for valid input. In the future it might be nice to allow the ports of a switch to be rebound. SIGUSR1 was used for the signal, not USER1 as in spec because USER1 has no definition outside of the program. The main difficulty I had was with making the input parsing and logic of the network.

Testing and Results:
The programs were compared with the example inputs and the outputs were equivalent.

In addition I tested with my own trafficFiles. Tests with 1-3 switches were performed. They used variable amounts of seperators between traffic file lines as well. The test results are the same when the program makes the fifos or they already exist.

Acknowledgments:

What libraries do I need for the commands I'm thinking of/ poll and mkfifo
implementation help:
https://linux.die.net/

How to read a file:
http://www.cplusplus.com/reference/fstream/ifstream/open/

Send/Recieve packets/frames:
http://webdocs.cs.ualberta.ca/~c379/F18/379only/lab-messages.html
Copyright: CMPUT 379: U. of Alberta, Author: E. Elmallah

How to use poll guide:
http://www.unixguide.net/unix/programming/2.1.2.shtml

String token iteration:
https://stackoverflow.com/questions/236129/how-do-i-iterate-over-the-words-of-a-str
ing