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 and Sockets for communication, and I/O multiplexing for nonblocking I/O.

In this part sockets replace switch-controller communication and switches can be delayed

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

How to use:
Makefile commands (assignment spec):
- Executing ‘make’ produces the a3sdn executable file.
- Executing ‘make clean’ removes unneeded files produced in compilation.
- Executing ‘make tar’ produces the ‘submit.tar’ archive.

Running the program:
The program can be invoked as a controller using
"a3sdn cont nSwitch port"
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:
"a3sdn swi trafficFile [null|swj] [null|swk] IPlow-IPhigh serverAddress port"
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 should not be closed before the switches.

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 separate files to decrease recompile time and create logical code separation.

1. a3sdn
   a3sdn.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

7. util
      - miscellaneous functions (trims whitespace of strings)

6. parsers
    - parses traffic 'packets' and TCP connection CLI argurments
```

Project Status:
All functionality delivered to match example test files and specification. I originally did this assignment without changing any a2sdn device code and I liked it better that way. Switch behaviour on a closed controller was defined but the assignment specification required changes to existing code and that behaviour isn't required.

Assignment 2 feedback was also addressed. But I wish the marking would have been consistent.

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

Acknowledgments:

New: How to check if a socket was closed:
http://www.stefan.buettcher.org/cs/conn_closed.html

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-string