CMPUT 379
Hdesmara
Hannah Desmarais

CMPUT 379 Assignment 2 Report: a2w22

Objectives

This assignment is meant to teach a student how to properly use FIFO's to send information to different processes or instances of the same program, similar to a network or like tabs in a browser. It also provides experience polling for different types of information in order to make these named pipes nonblocking. Also, it helps the student learn how to parse arguments and simulate a network with TOR packet switches. Lastly, it provides further experience with signals, using them in a way that will not interrupt polling.

Design Overview

Important features:

- Commands are case sensitive.
- There is a limit of 7 packet switches that may be activated
- A limit of 100 rules may be added to a table
- Both switches and the forwarding table are stored in vectors to have more flexibility
- When a packet switch needs to ask the master switch for a new rule, it will pause the program until it receives an ADD packet to prevent lines after being read and processed before the rule is made. This prevents the master being asked for the same rule multiple times.
- I ended up having to make global variables for the counts of types of packets, the vector of switches, and the forwarding table as well as a flag for the alarm called alarmOn. I did this because you cannot pass arguments besides the signal number to the handler and I needed this information in the handler in order to successfully pause reading lines for the delay as well as printing the appropriate information upon receiving a signal.
- Master must be started first in order for the packet switches to work as intended.
- I used a fixed length struct containing all possible fields a packet would ever need to send information back and forth through FIFO's.
- I used a struct to hold the data for a rule in the forwarding table.

Project Status

As of now, my project is finished as described in the a2 pdf provided on eclass. Because I have a windows computer, it took some fiddling to be sure my FIFO's were created properly. Normally you can make a FIFO with a name like "fifo-0-2", however, I ended up having to add "/tmp/" before the name so it would be created properly on my machine as I prefer to write code on my laptop in ubuntu before testing it on the lab computers. Before submitting, I just removed the "/tmp/" from everywhere in my code so it would work with the FIFO's made for marking. I also struggled a little with how to properly open FIFO's in a way that would prevent me from getting a bad address error and ended up opening all reading FIFO's as soon as the program is activated for either the master or a packet and then only opened write when the neighbour wasn't null and I needed to send something. After sending it, I would immediately close the writing end upon success.

Testing and Results

The following was tested:

- Info for master: This was tested by starting different packet switches and using the info command to make sure they were printed out with the same information I had provided on the command line in the list.
- Info for packet switches: This was tested by starting master and using info after I had written code for HELLO and HELLO_ACK and ensuring that they got incremented properly. As well as making sure that rules were properly passed by manually typing a rule and sending to info. If it had the correct format and information in the forwarding table and the counts for packets received, then I knew it was working properly.
- HELLO: This was tested by starting multiple packet switches and using an info call in master to make sure that the number of HELLO's received by master matched how many packet switches I had started.
- HELLO_ACK: This was tested by starting master and multiple packet switches and ensuring that after sending a HELLO packet to master, that they would receive this packet back. This was tested by using print out statements to verify that a pkt struct had been received with this HELLO_ACK as the type. I also would use the info command to ensure that the type count for HELLO_ACK had been incremented.
- ASK: To test that a switch would send an ask to the master, I fed it information with an IP address which I knew neither of the attached switches had in their range. After giving it the information, I would use the info command to print out the type counts of both the master and the switch. If master had received an ASK packet from the switch, then ASK for both would equal one. Next, I fed a second switch information an IP address that was not in range of it's own but was in range of the first switch. If the ASK counts of both the switch and master increased, then it worked.
- ADD: To test the ADD function, I first used values for IP addresses which I knew wouldn't be captured by any switches. After the master received an ask and it can't find a switch with the appropriate IP range, it should just drop it. I would verify that master would send a drop rule back as an ADD packet by using an info command in the switch and the master program. For master, I would verify that the ADD was sent by checking it's type count. For the switch, I would first look at the type count to make sure it had received an ADD and then I would look at the forwarding table. If the table had a drop rule, I knew it worked. For forwarding rules, I used IP addresses which I knew it should forward and what direction it should go. I would then look at the forwarding rule and make sure it captures the IP range of the neighbour it would be sending a RELAYOUT to, and that the forward had the correct port listed for the direction. I did this with a multitude of cases to verify it worked as intended.
- RELAYOUT/RELAYIN: To test relay, I used values for the IP destination which I knew would have to be sent to another switch. If a correct forwarding rule was added, as tested in the ADD

rule section, I would use an info command to confirm that the direction was correct and used a print statement to test that the switch got the message to RELAYOUT which was also reflected in the type counts. For RELAYIN, I would check the switch that was meant to be receiving the RELAYOUT packet and ensure it did by using an info command to check that the type count also incremented there. I would also check that upon receiving the packet, that the switch checked if it belonged to it. If the first row in the forwarding table had a packet count of one, then I knew it had recognized that the packet was meant for itself and didn't attempt to relay any further or ask for a new rule from the master by checking the rules and the type counts. If the packet did not belong to it and there was already a rule for it in the table, I verified that it had sent the packet further along by checking the packet count for that row and the type counts of both RELAYIN and RELAYOUT had incremented. For rules it did not have for the provided IP destination, I checked that it would add the appropriate rule to the forwarding table and increment the appropriate type counts depending on what the rule came back as. I also tested this using multiple switches to make sure that if say, a packet belonged to switch 4 but came to 1, it would continue to relay until it reached the appropriate switch and then store the data.

- USER1 signal: To check that the signal worked, I would start a master switch and send it the signal from another terminal, if the information about the master was printed the same as if I had entered an info command and the program didn't terminate, I knew it had worked. I repeated this process for packet switches as well.
- DELAY: To check that delay was working properly, I would first check that the conversion of milliseconds in the data file to seconds for the timer was working right by printing out the value. I would then start two switch packets and make the first one pause at the beginning, meaning that when it had a line which should be relaying out to the second switch, it wouldn't be dropped because by the time that the timer was up, the second switch would be started. I would verify that it was able to forward the header packet by using the info command and confirming that it had relayed out to the second switch. I also checked that I could still issue commands to the program being delayed by making it delay for at least ten seconds and using the info command repeatedly and double checking that the table had not made rules for the other lines in the data file.

Acknowledgements

- For the split function, I consulted the example on eclass here: http://webdocs.cs.ualberta.ca/~cmput379/W22/379only/lab-tokenizing.html
- For more examples on FIFO's I consulted: https://www.geeksforgeeks.org/named-pipe-fifo-example-c-program/
- For information on the signal command: https://man7.org/linux/man-pages/man2/signal.2.html
- For signal handlers and SIGUSR1 I looked at: https://linuxhint.com/signal_handlers_c_programming_language/
- Also, this post on interrupted system calls: https://unix.stackexchange.com/questions/509375/what-is-interrupted-system-call