CMPUT 379 - Assignment #2 (10%)

Communicating Peer Processes using FIFOs (first draft)

Due: Tuesday, March 1, 2022, 09:00 PM
(electronic submission)

Objectives

This programming assignment is intended to give you experience in developing communicating peer processes that utilize signals for examining the progress of the running processes, FIFOs for communication, and I/O multiplexing for nonblocking I/O.

Problem Context

In recent years, many information technology companies have increased their reliance on data centers to provide the needed computational and communication services. A data center typically houses hundreds to tens of thousands hosts. Each host (also called a blade) includes CPU, memory, and disk storage. The blades are often stacked in racks with each rack holding 20 to 40 blades. At the top of each rack, there is a packet switch, referred to as the Top of Rack (TOR) switch, that interconnects the blades in the rack with each other, and provides connectivity with other blades in the data center and the Internet.

In this assignment, we consider a simplified hypothetical data center network where the TOR switches are connected to a special device, called a master switch. The architecture of the network uses a new network design paradigm, called *software-defined networking* where the TOR switches are assumed to be simple, fast, and inexpensive devices. The master switch collects and processes information that enable it to give routing information to the TOR switches.

Network Model

The assignment deals with a simplified network that includes the following devices.

- A *master* switch device that has $k \geq 1$ bidirectional communication ports to connect at most k TOR packet switches.
- A set of k TOR packet switches. Each switch has 4 bidirectional ports serving the following purposes.
  - Port 0 connects the switch to the master switch.
  - Ports 1 and 2: each port connects the switch to another peer packet switch. A *null* value assigned to a port indicates that the port is not connected to any switch.
  - Port 3 connects the switch to blades in the rack. Each blade has an IP address $\leq$ `MAXIP` $= 1000$. As the IP address associated with each blade is unique, this port is associated with a unique range of IP numbers, denoted `[IPlow,IPhigh]` (e.g., $[100, 200]$).

Network operation. During network operation, each packet switch receives data packets from either the blades in its rack (an input data file is used to specify such packets), or its neighbouring switches connected to ports 1 and 2 (if any exists).

In real life, each packet has a *header* part and a *data* part. The header part stores information like the packet type, the source IP address (`srcIP`) and the destination IP address (`destIP`). For simplicity in the assignment, only the header part of each packet is being exchanged among the switches. The function of the network is to route each packet header to the specified destination.

Forwarding tables. Each packet switch is a simple device that stores a *forwarding* table, where each row has the following fields:

srcIP_lo	srcIP_hi	destIP_lo	destIP_hi	actionType	actionVal	pktCount

Each row defines a *pattern-plus-action* rule, where the pattern is composed of the first four fields, and the action is specified by the last three fields. An incoming packet header (with specified `srcIP` and `destIP` addresses) matches a rule if `srcIP` $\in$ [`srcIP_lo`, `srcIP_hi`], and `destIP` $\in$ [`destIP_lo`, `destIP_hi`]. If a match is found, the switch applies the corresponding action as follows:

- If `actionType` $=$ `forward` then the packet is forwarded to the port number specified by `actionVal`, and the `pktCount` is incremented.
- If `actionType` $=$ `drop` then the packet is dropped from the switch, and the `pktCount` is incremented.

Initially (when a switch is rebooted), the forwarding table stores an initial rule. Subsequently, when a packet header arrives to the switch, the switch tries to find a matching rule in its forwarding table. If no match exists, the switch asks the master switch for a rule to add to the forwarding table and apply to the packet. Assume that the forwarding table in each switch is large enough to hold rules for processing at most 100 arriving packet headers.

Network topology and routing. The master switch is responsible for issuing rules to packet switches so as to enable successful routing of packets. To simplify its operation, we assume that packet switches are connected to form a simple path topology. In such a topology, port 1 of switch i is either assigned the null value, or connected to switch $i-1$ if $i > 0$. Similarly, port 2 of switch i is either assigned the null value, or connected to switch $i+1$ if $i < k-1$.

Program Specifications

In this assignment, you are asked to write a C/C++ program, called `a2w22`, that implements the transactions performed by our simple network. The program can be invoked to simulate a master switch using

```
% a2w22 master nSwitch
```

where `nSwitch` $\leq$ `MAX_NSW`$(= 7)$ is the number of switches in the network.

It can also be invoked to simulate a TOR packet switch using

```
% a2w22 pswi dataFile (null|pswj) (null|pswk) IPlow-IPhigh
```

for example, `"% a2w22 psw4 file1.dat null psw5 100-110"`. In this form, the program simulates packet switch number i by processing the commands in the specified `dataFile`. Port 1 (respectively, port 2) of packet switch i is connected to switch j (respectively, switch k). Either, or both, of these two switches may be `null`. Switch i handles traffic from hosts in the IP range `[IPlow-IPhigh]`.

Data transmissions among the switches and the master switch use FIFOs. Each FIFO is named `fifo-x-y` where `x` $\neq$ `y`, and `x` $=$ `0` (or, `y` $=$ `0`) for the master switch, and `x`, `y` $\in [$`1`, `MAX_NSW`$]$ for a packet switch. Thus, e.g., `sw3` sends data to the master switch on `fifo-3-0`.

Data File Format

Each packet switch reads its arriving data packets from a common `dataFile`. The file has a number of lines formatted as follows:

- A line can be empty
- A line that starts with `'#'` is a comment line
- A line of the form `"pswi srcIP destIP"` specifies that a packet with the specified source and destination IP addresses has reached port 3 of `pswi`. You may assume that the `srcIP` address lies within the range handled by the switch. Only `pswi` processes this packet header; other switches ignore the line.
- A line of the form `"pswi delay interval"` where `interval` is an integer in milliseconds, specifies that packet switch i should delay reading (and processing) the remaining part of the data file for the specified time interval. During this period, the switch should continue monitoring and processing keyboard commands and packets received from the attached devices. This features simulates delays in receiving packets from hosts served by the switch.

Note: The field separator used on each data line is composed of one, or more, space character(s).

Packet Types

Communication in the network uses messages stored in formatted packets. Each packet has a type, and carries a message (except `HELLO_ACK` packets). Your program should support at least the following packet types.

- `HELLO` and `HELLO_ACK`: When a packet switch starts, it sends a `HELLO` packet to the master switch. The carried message contains the switch number, the numbers of its neighbouring switches (if any), and the range of IP addresses served by the switch. Upon receiving a `HELLO` packet, the master switch updates its stored information about the switch, and replies with a packet of type `HELLO_ACK` (no carried message).

- `ASK` and `ADD`: When processing an incoming packet header (the header may be read from the data file, or forwarded to the packet switch by one of its neighbours), if a switch does not find a matching rule in its forwarding table, the switch sends an `ASK` packet to the master switch. The master switch replies with a rule stored in a packet of type `ADD`. The switch then stores and applies the received rule.

- `RELAY`: A switch may forward a received packet header to a neighbour (as instructed by a matching rule in the forwarding table). This information is passed to the neighbour in a `RELAY` packet.

The Master Switch Loop

When `a2w22` is invoked to simulate a master switch, the program uses I/O multiplexing (e.g., `select()` or `poll()`) to handle I/O from the keyboard, and the attached switches in a non-blocking manner. Each iteration of the main loop of the master switch performs the following steps:

1. Poll the keyboard for a user command. The user can issue one of the following commands.

   - **info**: The program writes the stored information about the attached switches that have sent a `HELLO` packet to the master switch. As well, for each transmitted or received packet type, the program writes an aggregate count of packets of this type handled by the master switch.
   - **exit**: The program writes the above information and exits.

2. Poll the incoming FIFOs from the attached switches. The master switch handles each incoming packet, as described in the Packet Types section.

In addition, upon receiving signal `USER1`, the master switch displays the information specified by the `info` command.

The Packet Switch Loop

When `a2w22` is invoked to simulate a packet switch, the program installs an initial rule in its forwarding table:

srcIP_lo	srcIP_hi	destIP_lo	destIP_hi	actionType	actionVal	pktCount
= 0	= MAXIP	= IPlow	= IPhigh	= FORWARD	= 3	= 0

The rule matches an arriving packet header with any possible `srcIP` $\leq$ `MAXIP` value, and a `destIP` $\in$ `[IPlow-IPhigh]`. Recall that the range `[IPlow-IPhigh]` is specified on the command line as addresses handled by the switch.

The program then uses I/O multiplexing to handle I/O from the keyboard, the master switch, and the attached switches in a nonblocking manner. Each iteration of the main loop performs the following steps:

1. Read and process a single line from the data file (if the EOF has not been reached yet). The switch ignores empty lines, comment lines, and lines specifying other handling switches. A packet header is considered *admitted* if the line specifies the current switch.

   Note: After reading all lines in the data file, the program continues to monitor and process keyboard commands, and the incoming packets from neighbouring devices.

2. Poll the keyboard for a user command. The user can issue one of the following commands.

   - **info**: The program writes all entries in the forwarding table, and for each transmitted or received packet type, the program writes an aggregate count of handled packets of this type.
   - **exit**: The program writes the above information and exits.

3. Poll the incoming FIFOs from the master switch and the attached switches. The switch handles each incoming packet, as described in the Packet Types section.

In addition, upon receiving signal `USER1`, the switch displays the information specified by the `info` command.

Examples. Example output will be posted on eClass.

More Details

1. This is an individual assignment. Do not work in groups.

2. Only standard include files and libraries provided when you compile the program using `gcc` or `g++` should be used (e.g., standard I/O library, math, and pthread libraries are allowed). In addition, code posted on eClass can be used.

3. Although many details about this assignment are given in this description, there are many other design decisions that are left for you to make. In such cases, you should make reasonable design decisions that do not contradict what we have said and do not significantly change the purpose of the assignment. Document such design decisions in your source code, and discuss them in your report. Of course, you may ask questions about this assignment (e.g., in the Discussion Forum) and we may choose to provide more information or provide some clarification. However, the basic requirements of this assignment will not change.

4. When developing and testing your program, **make sure you clean up all processes before you logout of a workstation.** Marks will be deducted for processes left on workstations.

Deliverables

1. All programs should compile and run on the lab machines (e.g., ug[00 to 34].cs.ualberta.ca).

2. Make sure your programs compile and run in a fresh directory.

3. Your work (including a Makefile) should be combined into a single tar archive 'submit.tar' or 'submit.tar.gz'.

   (a) Executing 'make' should produce the `a2w22` executable file.

   (b) Executing 'make clean' should remove unneeded files produced in compilation.

   (c) Executing 'make tar' should produce the tar archive.

   (d) Your code should include suitable internal documentation of the key functions. If you use code from the textbooks, or code posted on eclass, acknowledge the use of the code in the internal documentation. Make sure to place such acknowledgments in close proximity of the code used.

4. Typeset a project report (e.g., one to three pages either in HTML or PDF) with the following (minimal set of) sections:

   – **Objectives:** state the project objectives and value from your point of view (which may be different from the one mentioned above)

   – **Design Overview:** highlight in point-form the important features of your design

   – **Project Status:** describe the status of your project; mention difficulties encountered in the implementation

   – **Testing and Results:** comment on how you tested your implementation

   – **Acknowledgments:** acknowledge sources of assistance

5. Upload your tar archive using the **Assignment #2 submission/feedback** links on the course's web page. Late submission is available for 24 hours for a penalty of 10% of the points assigned to the phase.

6. It is strongly suggested that you **submit early and submit often**. Only your **last successful submission** will be used for grading.

Marking

Roughly speaking, the breakdown of marks is as follows:

20% : successful compilation of reasonably complete program that is: modular, logically organized, easy to read and understand, includes error checking after important function calls, and acknowledges code used from the textbooks or the posted lab material

05% : ease of managing the project using the makefile

65% : correctness of implementing the master switch and the switch modules and displaying the required information using keyboard commands and signals.

10% : quality of the information provided in the project report