**UNX511/DSP912 – Assignment 1: A Network Monitor**

**Due: Monday, March 25th, 11:59PM**

**Business Case** (Marketing, Project Manager, Senior Executives)

Your job will be to create a network monitor, which monitors the performance of all network interface devices. Monitoring network interfaces can help in debugging system problems as well as in enhancing system performance through a central intelligent networking device.

Currently on our virtual machine, we have two network interfaces: **ens33** and **lo**. On other machines such as routers, there may be as many as 64 network interfaces, some of them running in the order of 10GBits/second. For routers and other networking equipment, it is crucial to know the performance of each interface and to control each interface, therefore we are going to create a network monitor.

**Requirements** (Marketing, Project Manager, Project Lead)

1. Our network monitor must be able to monitor any set of network interfaces requested by the user.
2. The statistics for each network interface can be found under Linux’s **/sys** directory, specifically **/sys/class/net/**<interface-name>. The **/sys** directory is another directory like **/proc** and **/dev** which can be accessed to set or obtain information about the kernel's view of the system.

The statistics to be monitored are:

The operating state – **operstate**

The up-count (number of times the interface has been up) – **carrier\_up\_count**

The down-count (number of times the interface has been down) – **carrier\_down\_count**

The number of received bytes – **rx\_bytes**

The number of dropped received bytes – **rx\_dropped**

The number of erroneous received bytes – **rx\_errors**

The number of received packets – **rx\_packets**

The number of transmitted bytes – **tx\_bytes**

The number of dropped transmitted bytes – **tx\_dropped**

The number of erroneous transmitted bytes – **tx\_errors**

The number of transmitted packets – **tx\_packets**

1. If an interface goes down (see **operstate**) then your network monitor must set it back up.
2. Your network monitor must shut down gracefully. That means it must intercept a **ctrl-C** to shutdown the program. The shutdown might include the following – close file connections, deallocate memory, etc…
3. The network monitor will act synchronously. It will not instantly react to any changes in the statistics. Rather, it will poll the statistics every one second.
4. Your network monitor will print out the following once every second for each network interface:

**Interface:**<interface-name> **state:**<state> **up\_count:**<up-count> **down\_count:**<down-count>

**rx\_bytes:**<rx-bytes> **rx\_dropped:**<rx-dropped> **rx\_errors:**<rx-errors> **rx\_packets:**<rx-packets>

**tx\_bytes:**<tx-bytes> **tx\_dropped:**<tx-dropped> **tx\_errors:**<tx-errors> **tx\_packets:**<tx-packets>

**Design** (Project Lead, Senior Engineers)

There are two parts to the design – the interface monitor and the network monitor itself. There may be one or more interface monitors but there is only one network monitor.

**The Interface Monitor**

Each interface will have a process dedicated to it. This is called the interface monitor. For instance, if there are three network interfaces on a Linux machine, three instances of the interface monitor process will be running. Each interface monitor will:

* Communicate with the network monitor through a temporary socket file in the **/tmp** directory.
* Inform the network monitor when it is ready to start monitoring.
* Monitor and print out the statistics for a given interface as per instructed by the network monitor.
* Report to the network monitor if its interface does down.
* Be able to set an interface link up as per instructed by the network monitor.
* Perform a controlled shutdown as per instructed by the network monitor by closing the socket connection to the network monitor then exiting.

**The Network Monitor**

The network monitor controls all interface monitors running on the system. In a [software defined network](https://www.ciena.com/insights/what-is/What-Is-SDN.html), it is common to centralize all intelligence in one spot in order to coordinate the operations of all network devices. In our case, intelligence is centralized in the network monitor and the work is done by the interface monitors. The network monitor will:

* Query the user for the number of interfaces and the name of each interface.
* It will create and bind a socket to a temporary socket file in the **/tmp** directory.
* It will maintain separate socket connections to each interface monitor.
* Once an interface is ready, the network monitor will instruct the interface monitor to monitor and print out the statistics for its interface once every second.
* If the interface monitor indicates that its link is down, the network monitor will instruct the interface monitor to put the link back up.
* On shutdown, the network monitor will intercept the SIGINT interrupt (ctrl-C, kill -2), send a shut down message to each interface monitor, close all connections, then exit.

Interface Monitor

Child 0

intf 0

Interface Monitor

Child 1

Network

Monitor

intf 1

Interface Monitor

Child 2

intf 2

Interface Monitor

Child (n-1)

intf (n-1)

**Implementation** (Junior and Senior Engineers)

* The network monitor will communicate with each of its interface monitors through a socket file in the **/tmp** directory.
  + The network monitor will create and bind a socket to a temporary socket file in the **/tmp** directory.
  + The network monitor will listen for an interface monitor to connect.
  + The interface monitor will create a socket and attempt to connect to the socket file in the **/tmp** directory.
  + The network monitor will accept the connection and fork off a child to communicate with that particular interface monitor.
  + The network monitor will then return to the accept. If another interface monitor attempts to connect to the socket file in the **/tmp** directory, the network monitor will accept the connection and fork off another child to communicate with that particular interface monitor.
* The communications between the network monitor and its interface monitors is synchronous, in that whenever the network monitor writes to an interface monitor, it waits to read something back before writing again.
* The protocol between the network monitor and the interface monitor is as follows:
  + Once the network monitor has received a set of interfaces from the user, the network monitor starts up each interface monitor as follows:

**./intfMonitor <interface-name>**

This can be accomplished with the system command which simulates a command from the command line:

**system(“./intfMonitor <interface-name>”);**

* + The network monitor waits for a connection request from each interface monitor.
  + The interface monitor writes “Ready” to the network monitor through a socket.
  + The network monitor writes “Monitor” to the interface monitor to monitor and print out the statistics of a given interface.
  + When the interface monitor is done monitoring statistics, it will reply with either “Done” or “Link Down”.
  + If the interface monitor replies with “Done”, the network monitor will sleep for one second.
  + If the interface monitor replies with “Link Down”, the network monitor will send “Set Link Up” to the interface monitor and then sleep for one second.
  + If the interface monitor receives “Set Link Up” from the network monitor, it will execute an IOCTL command to set the link back up.

(see [linux-programmatically-up-down-an-interface-kernel](https://stackoverflow.com/questions/5858655/linux-programmatically-up-down-an-interface-kernel))

* + If the network monitor receives a SIGINT (ctrl-C, kill -2), it will send a “Shut Down” message to each interface monitor, close all of its connections, and exit.
  + If the interface monitor receives a “Shut Down” message from the network monitor, it will close connections and exit.
* Monitoring statistics for each interface is quite simple. Each interface monitor simply has to open each file in the **/sys/class/net/**<interface-name> directory and read one value. No parsing is necessary.

**Testing and Rework** (Junior and Senior Engineers, Product Support)

* You may wish to insert a DEBUG flag in your Makefile and in your code to perform debug printf’s/cout’s while debugging. If you make your code with your debug flag, then code within the following blocks will be executed:

**#ifdef DEBUG**

**cout<<”DEBUG: …”<<endl**

**#endif**

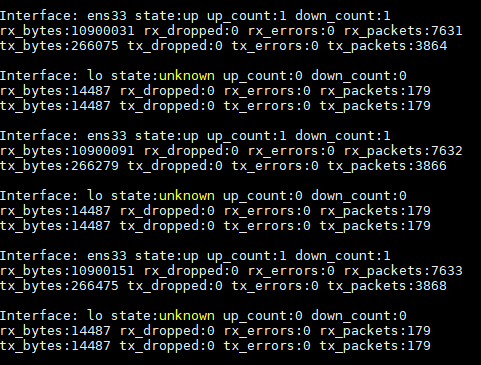
When you are satisfied that your code is working properly, you can build your code without the debug flag.

* When you run your debug monitor, compare the statistics that is printed out with the values in the **/sys/class/net/**<interface-name> directory.
* When you run your debug monitor, compare the statistics that is printed out with those shown with $ **ifconfig**
* To test the link down/link up feature of your interface monitor, put a link down (for example **ens33**) with the following command from the command line:

$ **sudo ip link set ens33 down**

Your interface monitor should indicate that the link is down temporarily, then indicate when it has come back up. Your up\_count and down\_count should each increment by one as well.

* A correctly running network monitor will look something like:



**Questions**

1. In a few sentences, describe the general architecture of your interface monitor.
2. In a few sentences, describe the general architecture of your network monitor.
3. Could the interface monitor and network monitor all be contained within one process with no child processes at all?
4. Could the interface monitor and network monitor all be contained within one process with no child processes at all? This time assume 128 network interfaces running at several Giga-bits per second, which require a polling interval of one millisecond per interface.
5. What is a software defined network?
6. The network monitor handles multiple socket connections through forking a child to handle each connection. Is there another way for the network monitor to handle multiple socket connections?

**Timeline**

**Week 1:** Complete and test the interface monitor as a stand-alone unit.

**Week 2:** Complete the network monitor with socket interface to the interface monitor.

**Week 3:** Test and Rework. Answer Questions. Submit Assignment.

**Marking Rubric**

Assignment 1 is worth 15% of your final grade and as such is marked out of 15 as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Does not meet expectations** | **Satisfactory** | **Good** | **Exceeds Expectations** |
| **Interface Monitor (3 marks)** | Does not meet requirements | Meets the most important requirements | Meets all requirements with minor errors | Meets all requirements with no errors |
| **Network Monitor (3 marks)** | Does not meet requirements | Meets the most important requirements | Meets all requirements with minor errors | Meets all requirements with no errors |
| **Documentation**  **(3 marks)** | Does not contain documentation. | Contains header documentation for either all files or for all functions within each file | Contains header documentation for all files and for most functions within each file. | Contains header documentation for all files and for all functions within each file.  Documents unclear code. |
| **Questions**  **(6 marks)** | Answers no question correctly  (0 marks) | Answers some questions correctly  (1-2 marks) | Answers most questions correctly  (3-4 marks) | Answers all Questions correctly  (5-6 marks) |

**Submission**

Please email all source code and answers to questions to:

**miguel.watler@senecacollege.ca**

Your questions can be submitted in a separate document or embedded within your source code.

**Late Policy**

You will be docked 10% if your assignment is submitted 1-4 days late.

You will be docked 20% if your assignment is submitted 5-6 days late.

You will be docked 30% if your assignment is submitted 7-8 days late.

You will be docked 40% if your assignment is submitted 9-10 days late.

You will be docked 50% if your assignment is submitted over 10 days late.