February 20, 2022

ECE428 Machine Programming 0

Due: 11:59 p.m. on Monday, March 6th, 2023

The objective of this MP0 is to create a simple network application. You can receive up to 5 marks for this MP, if you work yourself or with another student, and 2.5 (2%) if you work in a group of 3 (4) students. You can use any programming language, however, please contact the TA first, if you want to use language other than C/C++, Java or Python.

Task 1

Event logging is fundamental for monitoring the operation and performance of processes in distributed systems. It enables debugging and diagnostics, and creates understanding of what is happening. In this MP, the task is to implement a simple logger that receives messages from multiple network nodes, and store them in a centralized log. This will be useful in the next MP.

The network node should use the Python script `generator.py`, which generates messages for random events. The generator script can be given two input arguments. The first argument defines the average rate of generated events/messages, and the second argument limits the total number of generated events/messages. For instance, the rate 0.1 events/second means that, on average, there is one event every ten seconds. The messages are represented by random string of characters preceded by timestamps. The timestamps are expressed as the fractional number of seconds since January 1st 1970 (Unix standard).

The centralized logger should allow other network nodes to send their messages to the logger using TCP protocol on a port specified as a command-line argument. For instance, the command `logger 1234` starts the logger, and it then accepts the node connections and their messages at port 1234. The logger should write into a log file the node name and its connection time, its messages, and when the node disconnects. A sample log file may look like this:

```
1579666871.892629 - node1 connected
1579666871.9974 node1
2c7d235d2dc1ceee78d5521fae1e53c21f216af3b6685a37d3263137a95e116b
1579666872.10252 node1
ad0d8bb72c4fb74ee9095c1cac3e11e7f56b180eb19fb1e01faded0feff8984a
1579666872.2044811 - node2 connected
1579666872.307186 node1
38e88dd2999db43368662bccfb03587280cbfd51208aed27bd81462b0404508f
1579666872.409765 node2
c7da3c0a7135342ff80a111f36b3d32f5b80c4cecd536caf96930ae5dfc6b5ed
1579666872.514535 - node2 disconnected
1579666872.614976 node1
9e30f271fe6e65416eae5b9787e3d42ac406a260160e18692f379c4fa19e7a27
1579666872.7168908 - node3 connected
1579666872.820116 node3
17fb03ee5ffe9431b0c44c89995c3d82060267c96a2e4954564bd456f653b4c7
[...]
```

The first field is the time of the event (when it was generated by the node); the second field is the name of the node that generated the event. The remainder of the line is the event itself.

The network nodes can be implemented as receiving the messages/events from the standard input, and their main task is to send the messages to the centralized logger:

```
% python3 -u generator.py 0.1 | node node1 127.0.0.1 1234
```

The first argument of command `node` is the node name, and the second and the third arguments are the IP address and the port number of the centralized logger. Note that if the port number specified here is different from the one that was used to start the logger, no messages can be received by the logger.

The overall procedure is to first start the logger, and have it listening for messages at a given port. Then launch several network nodes locally on the same machine as the logger using standard IP 127.0.0.1 for a local host. Multiple nodes can be launched in different sub-shells using '&', or in different terminal shells.

Task 2

Once the network (distributed) system is working, we can evaluate the produced log file. For instance, we can track the following two metrics: (1) the time difference or delay between the time the event was generated, and when it was recorded in the log file, (2) the network bandwidth used by the logger. For instance, the logger can use the current time when the messages are being written to a log file, and also count the number of bytes received in all messages from all the network nodes over time. The metric values can be directly stored by the logger into a separate file, or send to a standard output, and redirected into a file by the shell. For instance, the output file with the two metrics may look like this:

```
# node, delay [sec], bandwidth [bytes/sec]
A, 0.532, 835.333
A, 0.388, 953.237
B, 0.120, 1340.640
A, 0.622, 380.333
[...]
```

More specifically, perform the following experiment. Setup three nodes A, B and C. The nodes B and C each generate 100 messages at the fixed average rate $R_B=1.0$ and $R_C=2.0$, respectively. The node A also generates 100 messages at the fixed average rate $R_A=\{ 0.1, 0.2, \ldots, 1.0 \}$. Plot the average, maximum, and minimum delays and the bandwidth used by the logger as a function of R_A. Comment on the plots whether any interesting behavior can be observed.

Submission

The MP assignment should be submitted via Gitlab on your main branch. Please follow the tutorial on Blackboard for the general submission tutorial. There is a 2% penalty per each starting hour if the MP has been submitted after the stated deadline. When the marks for the MP are released, you have 48 hours to request the regarding. The written regarding request must be submitted by email directly to the TA, explaining the reasons:

For this MP0, please submit:

- Your codebase
- A typed report that includes:
    - Outline of your codebase and general design choices
    - Instructions for running your codebase. It should demonstrate that the tasks defined in MP0 were accomplished.
    - Any packages or libraries that are required to run your code. Please also include a Makefile if you are using complied languages (C++, Rust, Golang, etc.).
    - Description on how to measure the delay and bandwidth defined in Task 2 above.

    - Name, student ID and email of your teammates.
- It is highly recommended to include a test suite for both your development experience and functionality demonstration.
- Plots for the average, maximum, minimum delay and bandwidth as defined above.

Grading Rubric

- 10% for clear instructions on how to build and run your code
- 40% for functionality
- 40% for collecting delay and bandwidth measurements
- 10% for producing readable plots with labeled axes and units