CPSC 559 – W2021 – Project Iteration 2

Contents

System Description

Now that you can communicate with the registry process and receive an initial list of peers, you can start working on the functionality for each peer.

Each peer will participate in a Twitter-like system. Your process should allow a user to enter a snippet of text content, which will be shared will all known peers in the system. Your process should also show all snippets to the user, in happens-before order, on a regular basis.

Requirements for a Peer Process

You may use any programming language you wish to create your peer process. The instructor and TAs will indicate which languages they can provide support for. Make sure to choose one of those languages if you want support from the TA or instructor. (If you do not need any support in coding, feel free to use any language you wish. Do make sure you document your code carefully to improve the readability of your code, even for those that are not familiar with the language you choose.)

All communication between processes in the system will be using Unicode characters. **Make sure to convert numbers to strings (or a sequence of Unicode characters) before sending them over the network.**

Your peer process must keep track of the list of locations of peers in the system that it is aware of. For this iteration, it will only contain the initial list of peers that you will receive from the registry. Use your code from iteration 1 to get an initial list of peers from the registry. There are some small modifications required in the communication with the registry. See the next section for details on the required modifications.

Once the initial peer list is received from the registry, your peer process should start communicating with the end user and with other peers in the system. Peers collaborate with each other in managing their lists of peers in the system and the peer communicates with the end user to add new text snippets to the system and display all snippets it is aware of to the end user. To accomplish this, the peer should be running the following concurrently:

1. Send and receive UDP/IP packets. The first four characters in the packet indicates the type of message is in the packet:
    a. 'stop' means that the entire system is shutting down and your process needs to initiate a shutdown procedure.
    b. 'snip' means that you are receiving a snippet that was initiated at another peer process in the system. (Optionally, you can send snippets initiated at your own peer to yourself as well.)
    c. 'peer' means that you are receiving information about a peer in the system. (This will help you maintain your list of peers.)
2. Collaborate with other peers in the system to maintain your list of peers. This means handling the 'peer' packets received to update your list of peers and intermittently sending 'peer' packets so other peers know you exist and are still alive. These 'peer' packets also give information about a third peer to help you grow your list of peers.
3. Maintain a sequence of all snippets in the system, such that they are ordered in a way that satisfies Lamport's happens-before ordering.

Each of these three responsibilities must run on their own thread (or similar construct) to ensure they can operate concurrently.

A registry and some test peers will always be running to help you test your code. The registry will initiate a 'stop' periodically to help you test this requirement as well. Alternatively, you can download the registry and run it locally to test the system shut down.

At minimum, you must be able to establish the initial connection with the registry and communicate with the registry for the shut-down procedure. If this does not work, **you will not be able to submit your code and final report and you do not qualify for any marks for this second iteration.**

Updated Registry Communication Protocol

Two updates are required for this iteration:

1) Location Request

```
<location request> ::= "get location"<newline>

<location response> ::= <peer>
```

For each peer, instead of using the location that was used for the TCP/IP connection with the registry, we will store the location where the peer will send/receive UDP/IP packets to/from. If you do not have a dedicated port you can use, then start the UDP server first before connecting with the registry. Then use the IP and port of this UDP server as your location.

2) Updated Report Request

```
<report request> ::= "get report"<newline>
<report response> ::=
        <peer list><peer list sources><peers recd><peers sent><snippet list>
<peers recd> ::= <numOfSinglePeerSources><newline><single recd peers>
<peers sent> ::= <numOfSends><newline><single sent peers>
<snippet list> ::= <numOfSnippets><newline><snippets>
<single recd peers> ::= <single recd>|<single recd><single recd peers>
<single sent peer> ::= <single sent>|<single sent><single sent peers>
<single recd> ::= <source peer><space><received peer><space><date><newline>
<source peer> ::= <peer2>
<received peer> ::= <peer2>
<single sent> ::= <sent to peer><space><peer sent><space><date><newline>
<sent to peer> ::= <peer2>
<peer sent> ::= <peer2>
<peer2> ::= <ip><colon><port>
<snippets> ::= <snippet>|<snippet><snippets>
<snippet> ::= <timestamp><space><content><space><source peer><newline>
<timestamp> ::= <num>
<content> ::= .*
<numOfSinglePeerSources> ::= <num>
<numOfSends> ::= <num>
<numOfSnippets> ::= <num>
```

Explanation: once the connection is established, the registry sends the string 'get report' followed by a new line character. The registry then waits to receive, in order,

1. your current list of peers (same as iteration 1), followed by

2. all lists that you have received (same as iteration 1), followed by
3. all peers that your received via a UDP/IP message, followed by
4. all peers that you sent via a UDP/IP message, followed by
5. all snippets that you have received, in timestamp order.

For point 2, you will only have received one list of peers, namely the list you received from the registry when you joined the system.

For example: the following is a possible response to a 'get report' request.

Report	***Explanation***

4 136.159.5.25:61162 136.159.5.25:28345 136.159.5.25:54699 136.159.5.25:64057	The reporting peer has a list of 4 peers
1 136.159.5.27:55921 2020-11-30 12:49:25 1 136.159.5.25:64057	There was one source that provided a list of peers (that would have been the registry) The date this list was received There was one peers in the list The one peer that was received
9 136.159.5.25:64057 136.159.5.25:64057 2020-11-30 12:49:25 136.159.5.25:64057 136.159.5.25:64057 2020-11-30 12:49:35 136.159.5.25:61162 136.159.5.25:61162 2020-11-30 12:49:36 136.159.5.25:28345 136.159.5.25:64057 2020-11-30 12:49:37 136.159.5.25:54699 136.159.5.25:64057 2020-11-30 12:49:37 136.159.5.25:64057 136.159.5.25:64057 2020-11-30 12:49:45 136.159.5.25:61162 136.159.5.25:64057 2020-11-30 12:49:46 136.159.5.25:28345 136.159.5.25:61162 2020-11-30 12:49:47 136.159.5.25:54699 136.159.5.25:28345 2020-11-30 12:49:47	There were 9 peers that were received via a UDP
6 136.159.5.25:64057 136.159.5.25:64057 2020-11-30 12:49:25 136.159.5.25:64057 136.159.5.25:64057 2020-11-30 12:49:35 136.159.5.25:61162 136.159.5.25:64057 2020-11-30 12:49:45 136.159.5.25:28345 136.159.5.25:64057 2020-11-30 12:49:45 136.159.5.25:54699 136.159.5.25:64057 2020-11-30 12:49:45 136.159.5.25:64057 136.159.5.25:64057 2020-11-30 12:49:45	There were 6 peer messages sent via UPD
9 3 test0 test message 1 136.159.5.25:61162 4 my test 136.159.5.25:64057 9 test1 test message 1 136.159.5.25:28345 12 test2 test message 1 136.159.5.25:54699 15 test0 test message 2 136.159.5.25:61162 18 test1 test message 2 136.159.5.25:28345 19 another test 136.159.5.25:64057 24 test2 test message 2 136.159.5.25:54699 27 test0 test message 3 136.159.5.25:61162	This peer is aware of 9 snippets in the system Time stamp is 3 for this snippet and originated from 136.159.5.~~1~~25 at port 61162 and content is 'test~~1~~0 test message 1'

Group management requirements (*peer* UDP/IP messages)

The list your peer receives from the registry is only a partial list of peers in the system. This list will also not contain any peers that joined after you did. Peer processes are expected to collaborate in building a more complete peer list. This is accomplished by sending and receiving *peer* UDP/IP messages. You should send *peer*

messages at regular intervals to all peers in your list. If you have not received any messages from a peer in your list for some time, assume that this peer has left the system and stop sending messages to that peer. (Do keep this peer in your list for the report.)

A peer message should have the following format:

`<peer msg> ::= peer<peer info>`

`<peer info> ::= <IP><colon><port>`

The peer in the message is one of the peers in the list of the sender. Make sure to record the sender info, the info of the peer in the message and the date the UDP message was received for the report.

User interface and snippets requirements (*snip* UDP/IP messages)

For this iteration create a simple user interface that allows the user to add snippets of text to the system and that displays all snippets that are in the system. This interface can be a simple text-based interface using the console or it can be a graphical user interface.

When the user enters a snippet, send this snippet to all peers in the system using the following format:

`<snippet msg> ::= snip<snippet timestamp><space><content>`

Use Lamport's logical clock to ensure that snippet timestamps will satisfy a happens-before order. See section 6.2 in the text for details on Lamport's logical clocks and an algorithm for attaching a logical clock timestamp to each snippet. (Note that the sender is always the one assigning a timestamp to a snippet.)

For the report, make sure to store the timestamp, content, time received and sender of each snippet.

Shutting down system requirements (*stop* UDP/IP messages)

The last UPD/IP message that your peer is required to handle is a *stop* message. It will have the following format:

`<snippet msg> ::= stop`

When you receive it, let the user know the system is shutting down and end all threads (or similar constructs) that are running (the UDP server, the peer list manager and the snippet interface). Connect with the registry again and respond to requests from the registry. One of these requests will be a report on all activity of your peer since it started running. Once communication with the registry is completed, your peer should halt.

Testing Your Solution

Before the iteration due date, a Registry server will be running (mostly) continuously. By default, it will be running at 136.159.5.22:55921. There will also always be at least four peer processes running that will generate random snippets and will participate in the group management as required. The code for the Registry and the automated test peers will be provided if you want to run them locally before joining the rest of the class to test your peer process. If the registry is not running at the default location or the location last communicated on the

discussion board, post a note on the discussion board and I will get the Registry restarted as quickly as possible. (Note that the registry for iteration 2 will only start running after the iteration 1 due date has passed.)

This test registry will send requests in the following order when you first connect to the registry (as long as the system is not shutting down):

1. 'get team name'
2. 'get code'
3. 'receive peers'
4. 'close'

When the system is shutting down, you will receive the following requests from the registry:

1. 'get team name'
2. 'get code'
3. 'get report'
4. 'close'

The Registry that will be used to submit your solution will be the same as the one used for testing, but it will be restarted before submission starts.

For your convenience, the source code of this test registry is available on D2L.

Submission

This second iteration is due on Friday March 4 at 4pm. There are three requirements for submission:

1. Class diagram of your solution to the D2L dropbox for this iteration.
2. A video that gives a brief explanation of your implementation and shows your peer in action. The length of your video should be between 2 and 7 minutes. Only the first 7 minutes of your video will be viewed when grading. Use the same D2L dropbox to either upload the video or provide a link to your video.
3. Running your peer on Friday March 4 between 3 pm and 4pm.

Grading will be based on:

1. Your ability to connect with the submission registry server and your ability to send your source code and report when required and requested. If this portion is not functioning, your submission will NOT be graded.
2. The accuracy of the report that your process sends to the registry sever during shut down.
3. The quality of the code. (Design, legibility and documentation)
    a. If no class diagram is provided, there is no marks for design.
    b. If the video does not give a brief overview of the design and organization of the code, there are no marks available for design.
4. The ability for an end user to send snippets and read snippets. If the video does not show this functionality clearly, no marks are available for the interface.

Collaboration Opportunities and Limitations

You may continue with the same partner, work with a different partner, or work alone for this iteration. Each team must have a unique team name. Please contact the instructor with a preferred team name and the list of team members.

No other individuals should work directly on the code you write, except for the instructor and TAs. Do not send your entire code base to any individuals nor make it available in a public repository, a discussion board or any other forums.

You may use any other resources you find useful. You may ask questions in the course discussion boards and any public discussion boards. Do make sure you cite any sources you use and any suggestions you received on discussion boards and other forums. An example on citing code using code documentation can be found at: https://runestone.academy/runestone/books/published/StudentCSP/CSPCreativity/referencing.html Do make it clear exactly which of your submitted code the citation relates to.

If you find an algorithm at a source other than the materials provided for class, make sure you cite this as well in the code documentation.

If you are not sure if a certain level of help or collaboration from individuals outside the team is allowed, make sure to contact the instructor for clarification.