Monday, 20 November 2017

EE 324, Programming Assignment #4

Implementing a peer-to-peer network application

1 Overview

You need to implement a KaZaA style p2p application, which hires a super-node approach to improve the performance of searching. First, you need to implement two types of p2p applications; (i) child node and (ii) super node. Then, you should check if implemented nodes will conduct required functions correctly.

2 Important Notes

- All connections should be handled with the I/O multiplexing – use ***select()*** call
- Accepting new clients and handling all send/recv (or other related) functions should be managed by select, which means that you should implement both a child node and a super node with select() call
- Each node (child or super) can be implemented with a single-thread (if you can) or multiple threads (if you want) → your design choice
- Each network transaction should be handled no more than 2 seconds (except file downloading)
- Queues, domain sockets, shared memory… you can choose any data structure and any scheduling method that you want → again, your design choice
- But, you should not be quite slow (no more than 2 seconds)

3 Specification: Child Node

3.1 Content handling

a. It should scan a local directory (/data) to catch which files are existing in there. In this case, TA will create a directory /data into the directory where your binary is located (e.g., if your binary is located at */home/test/PA4/bin/your_binary* then, TA will create the /data directory under the location ➔ */home/test/PA4/bin/data*). If there are three files in /data (e.g., a.txt, b.txt, and c.txt), your application should catch the name of each file and the size of each file

b. Scanned information of each file (name and size) should be delivered to a super node

3.2 Connection

a. A child node should connect to (initiator) a pre-defined super-node (in this case, we assume that each child node already has the information of a super-node). At this time, each child node only connects one super-node

b. Note. You need to provide a way of setting the information of a super node (IP and port)

c. A child node also provides a way of receiving a connection request from other peers (which means that a child node should support the functionality of both client and server)

d. If other child nodes ask a file, a child node should deliver the required content (as a network server)

e. A child node will ask a super node to get the information of certain file. Simply considering a situation that a p2p node downloads a file. In this case, a child node will send a query to a super node to know the location of a file. The downloaded file should be located in 'download' directory. (e.g., if your binary is located at */home/test/PA4/bin/your_binary*, then TA will create the /download directory under the location ➔ */home/test/PA4/bin/download*)

4 Specification: Super Node

4.1 Content handling

a. Child node information should be maintained (child IP, port for service)

b. Delivered information from each connected child node (file info and child node info) should be maintained and shared with other super nodes. You should design your own data structure (hash table?) to hold the information.

4.2 Connection

a. A super node should handle multiple connection requests from child nodes.

b. A super node should make a connection to other super node (in this assignment, there are only two super nodes, and you can choose any way to make a connection between them).

5 Protocol

Total Length (4 byte)
ID (4 byte)
MSG Type (4 byte)
Payload (variable)

Figure 1. Protocol layout

Figure 1 denotes the protocol for this p2p application, and each child and super node should understand this protocol.

- **Total length**: all payload length (application data including this header)
- **ID**: unique ID of a node
- **MSG type**
    - 0x00000010: HELLO from CHILD, when a child connects to a super node
    - 0x00000011: HELLO from SUPER, when a super node responds to a child connection trial
    - 0x00000012: HELLO from SUPER to SUPER, when a super node connects to another super node
    - 0x00000020: FILE INFO, when a child sends its file information to a super node
    - 0x00000021: FILE INFO RECV SUCCESS, when a super node answers to FILE INFO msg (successfully received)
    - 0x00000022: FILE INFO RECV ERROR, when a super node answers to FILE INFO msg (receive fail, only consider a case of not receiving all contents)
    - 0x00000030: SEARCH QUERY, when a child sends a query to a super node
    - 0x00000031: SEARCH ANS SUCCESS, when a super node returns search results successfully
    - 0x00000032: SEARCH ANS FAIL, when a super node returns failure for its query (no matched file)
    - 0x00000040: FILE REQ, when a child asks a file content to another child (NO multiple files, only one file at a time)
    - 0x00000041: FILE RES SUCCESS, when a child delivers file contents to a child that has requested the file
    - 0x00000042: FILE RES FAIL, when a child returns error to a child that has requested a file
    - 0x00000050: FILE INFO SHARE, when a super node delivers its file info to another super node
    - 0x00000051: FILE INFO SHARE SUCCESS, when a super node successfully receives file info
    - 0x00000052: FILE INFO SHARE ERROR, when a super node fails in receiving file info (receive fail, only consider a case of not receiving all contents)
- **Payload**
    - Design your own structure, and describe the structure clearly in your README FILE
    - When the MSG type is HELLO from CHILD
        - Child sends its ***port*** for receiving request from another child nodes to a super node
    - When the MSG type is FILE INFO
        - Child delivers ***file name*** and its ***size*** (consider multiple files)
    - When the MSG type is FILE INFO RECV SUCCESS
        - Super node delivers ***received file names***

- When the MSG type is SEARCH QUERY
  - Child sends a ***file name*** that it wants to download
- When the MSG type is SEARCH ANS SUCCESS
  - Super node sends a child node having the requested file (IP and port)
- When the MSG type is FILE REQ
  - Child should send a file name for downloading
- When the MSG type is FILE RES SUCCESS
  - Deliver a requested file content to a child node
- When the MSG type is FILE INFO SHARE
  - Deliver file list (name and its size) to another super node
- When the MSG type is FILE INFO SHARE SUCCESS
  - Return received file info

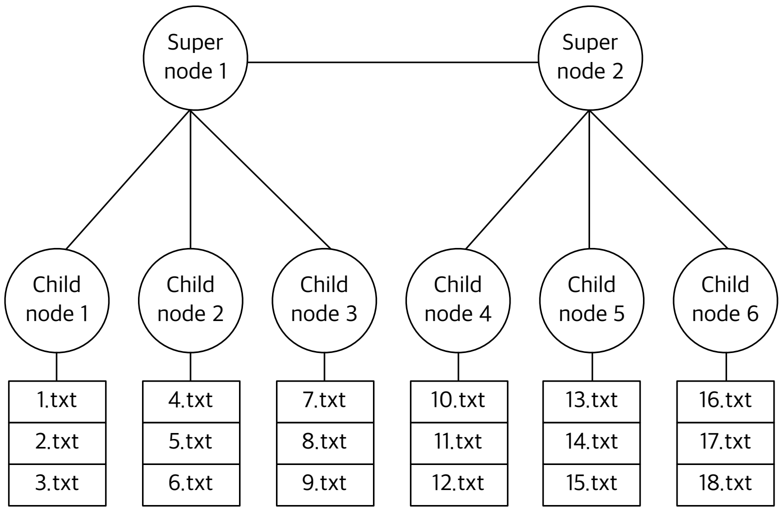

Figure 2. Overall test environment

6 Overall working scenario (see Figure 2, assuming no error case)

1) start super node 1
2) start super node 2 and send HELLO from SUPER to SUPER (from super node1 to super node2)
3) start child node1
   a. it first connects to super node1 (HELLO from CHILD) and (HELLO from SUPER)
   b. it checks local storage (/data) and notices 3 files. It sends file information to super node1 (FILE INFO) and (FILE INFO RECV SUCCESS)
4) super node1 shares this info with super node2
   a. FILE INFO SAHRE and FILE INFO SHARE SUCCESS
5) Other child nodes do the same operations (3) depending on each super node
   a. Super nodes also share information with each other
6) Then all child nodes have been connected to each super node and delivered their file information
7) Child node 2 asks super node1 to find 16.txt (SEARCH QUERY)
8) Super node1 returns its info (who has the 16.txt) to child node2 (SEARCH ANS SUCCESS)
9) Child node 2 sends a request to child node6 (FILE REQ)
10) Child node 6 returns the content to child node2 (FILE RES SUCCESS)

7 Evaluation

7.1 Preliminaries

- You **SHOULD** write detailed description of how you implemented your p2p applications in your document including your protocol design. Note that any unclear part will deduct your score. (You can use Korean)
- Each check for protocol will be done via packet monitoring, so you **SHOULD** be careful to describe your protocol design and specification in detail
- In this project, you **MUST** handle 'binary' data not only 'ascii' data. (e.g., you need to handle file contents including "\xad\xee\x00\xaa\xbb") Consider to use send, recv functions.
- You **SHOULD** implement error handling if timeout happens
  * (e.g.,) Each network transaction should be handled no more than 2 seconds (except file downloading)
- Each child node **MUST** receive user input to retrieve file as STDIN. For example, assume that super node is running on 127.0.0.1:11111, the command should be as follows:

```
$ ./child –p 12345 --s_ip 127.0.0.1 -s_port 11111
Input command:  get 12.txt asdf.txt
```

Figure 3. An example of a child node command

  * note that '-p' is to get the port number, '--s_ip' gets other super node IP address or Domain name, and '--s_port' gets other super node port number.

- Here, string "Input command" can be changed, but you need to keep the format "get [file_name] [destination_file_name]"
- Requested [file_name] **MUST** be saved as [destination_file_name] into "download" directory.
- TA will test as follows (you need to support below commands. All commands will be run with script file, so you MUST keep the format as below):
  * note that '$' represents shell command line

7.2 Test Case

1) Setup test environments

```
$ mkdir super1 super2 child1 child2 child3 child4 child5 child6
$ cp ./super super1
$ cp ./super super2
$ cp ./child child1
$ cp ./child child2
$ cp ./child child3
$ cp ./child child4
$ cp ./child child5
$ cp ./child child6
```

2) Start super node 1

  * note that '-p' is to get the port number

```
$ cd super1
$ ./super -p 11111
```

3) Start super node 2

  * note that '-p' is to get the port number, '--s_ip' gets other super node IP address or Domain name, and '--s_port' gets other super node port number.

```
$ cd super2
$ ./super -p 11112 --s_ip 127.0.0.1 --s_port 11111
```

- check if handshake with super node 1 is properly done.

4) For each child node from 1-3:

* note that ‘X’ represents the node index

```
$ cd childX
$ mkdir download
$ mkdir data
$ cp /tmp/X/* ./data
$ ./child -p 2222X --s_ip 127.0.0.1 --s_port 11111
```

- check if handshake with super node 1 is properly done
- check if super node 1 gets the list of file with right protocol
- check if super node 1 shares information with super node 2 with right protocol

5) For each child node from 4-6:

* note that ‘X’ represents the node index

```
$ cd childX
$ mkdir download
$ mkdir data
$ cp /tmp/X/* ./data
$ ./child -p 2222X --s_ip 127.0.0.1 --s_port 11112
```

- check if handshake with super node 1 is properly done
- check if super node 2 gets the list of file with right protocol
- check if super node 2 shares information with super node 1 with right protocol

6) Test to receive files randomly and simultaneously:

* note that ‘X’ represents the file index from 1 to 18.
- For each child node waiting to get stdin data, “get X.txt X.txt” string will be sent.
- check if search and request protocol is properly done.
- check if files are downloaded into each ‘download’ directory (check the hash sum of files).

7.3 Scoring

- documentation: 40 pts
- super node <-> super node handshake: 10 pts
- child <-> super node handshake: 10 pts
- child <-> super node file information: 20 pts
- super node <-> super node information sharing: 20 pts
- child -> super node file search: 20 pts
- child -> child file request: 30 pts

*** IMPORTANT NOTICE:** if you do not follow ‘MUST’ things, the test will fail and you will not get any point. KEEP THE FORMATS!

8 Instruction for Submission

- You will be submitting one taball file to the KLMS website.
- Create a tarball (tar.gz) with all the source codes, README, and executables.
- Tarball structure: (for program assignment 4)
- Create a folder called “p4"
- Put your source code in the folder named “p4/src”
- Put your executable(s) in the folder named “p4/bin”
- Put your “Makefile” in the root path - “p4/”
- Make sure that your Makefile correctly complies your source code
- readme.txt file goes to the root path - “p4/”
- No README, no points; let us know how to run your program
- install.sh goes to the root path as well – “p3/” (optional)
- But, if your program needs a third-party library, you should provide a script “install.sh”
- Then, compress the entire "p3" folder into a single tarball; the name will be **“P4_yourIDnumber.tar.gz”**
- Write the concise comments in the source code to understand your program.

! **IMPORTANT 1: Please strictly follow the above structure and name policy; if not, TAs will not evaluate your program.**

! **IMPORTANT 2: Please make sure that there is no compile and execution error. TAs will not give you any score in case of the problems.**

9 Test Environment

- Language: C or C++
- Test O/S: **Ubuntu 14.04.5 LTS (Trusty Tahr) 64bit**
- http://releases.ubuntu.com/14.04/
- If you need a VM, download using the following link.
- http://nss.kaist.ac.kr/ee324.ova
- username: ee324, password: ee324
- Import the VM using VMware or VirtualBox
- **NOTE: We will not consider your compilation and execution problems due to the different OS versions.**

10 Due Date

- **11:59 PM, Dec. 08, 2017 (Friday)**
- **No delay will be accepted!**

11 Questions?

- Please email TAs to ask questions.
- Junsik Seo (js0780@kaist.ac.kr)
- Jinwoo Kim (jinwoo.kim@kaist.ac.kr)